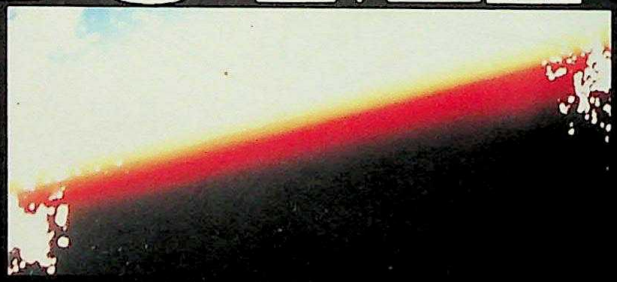
PC9801,PC98XA,N5200/05mkII,IBM5540,5550,5560,FM16β,if800/60
国際的標準データーベース言語

パソコンユーザーのための「MUMPS」100%活用書

BEGINNING MUMPS



ビギニングMUMPS

嶋 芳成 著

日本ソフトバンク

PC9801,PC98XA,N5200/05mkII,IBM5540,5550,5560,FM16β,if800/60
国際的標準データーベース言語

パソコンユーザーのための「MUMPS」100%活用書

BEGINNING MUMPS

ビギニングMUMPS

嶋 芳成 著

日本ソフトバンク

©1986 本書のプログラムを含むすべての内容は著作権法上の保護を受けております。著者，発行者の許諾を得ず，無断で複写，複製をすることは禁じられております。

MUMPSとの出会い

私が MUMPS に出会ったのは，ほぼ12年前日本に初めて MUMPS が導入されたころのことです。当時は，まだマイコンもなく，カナが使用できれば日本語処理とされ，漢字の利用できるコンピュータなど夢のまた夢でした。

ところがその後のコンピュータ技術の発展，そしてマイコン/パソコンの進歩と普及は目ざましい限りです。素人にも使えるシステムが無数に開発され，利用技術の情報が巷にあふれています。とくにリレーショナルデータベース（RDB）システムが， BASIC より生産性が高く，簡易言語より柔軟で拡張性が高いということで注目を集めています。 RDB を知らなければ時代遅れといった様相さえあります。

そのような技術の波の中でなぜ今 MUMPS なのでしょうか。一言でいえばバランス性です。 MUMPS と競合するプログラミング言語や，データベースシステムは無数にあります。しかし汎用性，拡張性，柔軟性，互換性，コンピュータの非専門家にも扱える使いやすさなど，バランスよく持っているという点では， MUMPS に匹敵するシステムは見あたりません。

たとえばC言語は，小回りがきき，互換性もよく，優れた言語ですが，一般ユーザー向けではありません。 COBOL も大きなコンピュータでは広く普及していますが，一般ユーザーには難しく，マイコンには重いと思われます。一方，マイコン用の RDB は入門しやすいのですが，それだけで何でもできるわけではなくメーカーが異なると互換性がありません。

MUMPS は，使いやすく強力なデータベースとして，従来ミニコン以上のコンピュータでしか利用できませんでしたが，このたびマイコン用のMUMPSがパソコンショップなどで入手できるようになり，普及の条件は整いました。

ところで，友人や同僚に MUMPS を勧めるときに，残念ながらこれだというテキストがありませんでした。

『標準 MUMPS 演習』（若井一朗訳）は非常に良くできた入門書です。しかし初心者にはこれでもまだ多過ぎ，難解という印象を与えるようです。

多くの人に MUMPS の講習をしながら，さらに簡単な解説書の必要性を感

じ，とうとう自分で書くことになりました。そこで勤務する久美愛病院の職員教育用に教科書を作ることを思い立ったのです。 MUMPS の文法は，ワープロと同じですべてを知っている必要はありません。知っている範囲内でそれなりの利用法があります。そこで文法は必要最小限にとどめ，MUMPS を使って問題解決することを目的としてプログラムの「定石」を重視することにしました。そうしてできあがったのがこの本です。

もちろん文字列操作機能は MUMPS の重要な能力ですが，それを知らなくても仕事には利用できます。しかしデータベース機能を知らないと，処理の結果を蓄えることができません。したがって文字列操作機能は別立てで触れることにしました。とにかくデータを蓄え，欲しいリストが作れるようにするということがこの本の主題です。

また同時にモジュラプログラミングにも慣れるように配慮しました。初心者は GOTO 命令を多用する傾向があり，後からそのプログラミングスタイルを矯正するのが大変です。京都大学医学部の平川顕名教授のアドバイスもあり，この本では GOTO 命令を使わないことにしました。

この本はしたがって，コンピュータについてまったくの初心者が対象です。しかし BASIC や RDB を使用しているかたが MUMPS的な発想を理解するためにもよい参考になるはずです。さらに従来の MUMPS ユーザーには，ワープロや作図システムと連携して MUMPS の世界を広げられるという，マイコンならではの自由度を理解する助けになると思います。

最後に，この本の出版にあたり草稿の不十分な点を指摘し，例題と実際の出力結果の違いを調べてくださった久美愛病院の橋本敦子氏，小倉喜子氏をはじめとする職員の皆さん，論理の飛躍を指摘してくださった高山赤十字病院の大久保恒正先生，名古屋大学医学部の錦見尚道先生，その他多くの方々の御助力に深謝いたします。MUMPS について御指導いただいたマンプスシステム研究所の若井一朗先生はじめ日本 MUMPS ユーザーズグループの皆さんにもこの場を借りて感謝の意を述べさせて頂きます。そして，出版に御尽力頂いた住友電気工業と日本ソフトバンク出版部にお礼申し上げます。

1986年 4 月

嶋　芳成

CONTENTS

Step 0 ようこそMUMPSワールドへ

Chapter 0 プロフィルMUMPS……10
MUMPSとは……10
1 MUMPSはデータベースシステム……10
2 MUMPS＝枝分かれ構造データベース……11
3 MUMPSは文章データを効率よく蓄える……13
4 MUMPS＝コンピュータ言語……13
5 MUMPSは標準化が進んでいる……14
6 MUMPSはさまざまのコンピュータで使用できる……15
7 MUMPSとネットワーク……16
8 MUMPSと人工知能……16
9 MUMPSと数値計算……16
10 MUMPSを何に使うか……17
「ビギニングMUMPS」の構成……18
1 本書の目的……18
2 本書の構成……18
3 読み始める前に……20

Step 1 スターティングMUMPS

Chapter 1 MUMPSを呼び出す……22
1.MUMPSを使うにはMS-DOSが必要……22
2.フロッピーディスクを準備する……22
3.MUMPSを呼び出すための手順……23
4.MUMPSの仕事を終了する……27
Chapter 2 MUMPSと対話する……29
1.デタラメにキーをタイプしてみる……29
2.簡単な計算をさせよう……30

3.計算手順を教え込む……33
4.ルーチンの内容を調べる……35
5.ルーチンを実行する……35
6.もう一度繰り返してみよう……36
7.表示を見やすくする……37
8.縦に並べて表示する……39
9.WRITE命令を一行で書く……40
10.ルーチンを修正する……42
Chapter 3 ルーチンについて……44
1.ルーチンの構造……44
2.ルーチンの作り方……46
3.ルーチンの内容を表示する……47
4.行を削除する……49
5.ルーチンの修正—行を入れ替える……51
6.ルーチンの修正—行を挿入する……52
7.もっと便利な修正方法……54
8.ルーチンを登録する……57
9.ルーチン名の付けかた……60
10.ディスク登録ルーチンの削除……61
11.登録ルーチンの名前を調べる……62
12.ルーチンヘッダーの慣例……63
13.Z命令について……66
Step2 プログラミングMUMPS
Chapter 4 データを記憶する……70
1.データのメモリと変数……70
2.SET命令を用いたルーチンを作る……73
3.変数の内容を調べる……75

4.データを削除する……76
5.いろいろな値を計算する……77
6.READ命令の代表的パターンを覚えよう……80
7.READ命令を用いた簡単なルーチン……81
Chapter 5 職員名簿を作る……85
1.キーボードでタイプしたデータを蓄える……85
2.多人数のデータを蓄える……86
3.変数に添え字をつける……87
4.添え字付き変数を利用してデータを蓄える……91
5.入力したデータを書き出すルーチン……92
6.データをリストアップする……93
7.データを表示する位置を制御する……97
8.命令を複数並べる……99
9.女性のみのリストアップ……100
10.いろいろな条件のリスト……104
11.名前による検索……106
12.名前の一部で検索する……107
Chapter 6 データを永久保存する……109
1.グローバル変数……109
2.グローバル変数を用いたデータ入力ルーチン……112
3.添え字を複数つける……113
4.データのリスト／検索ルーチンの訂正……116
5.あらゆるグローバル変数に使えるグローバルリスト
ユーティリティ……119
6.ルーチンを呼び出すときの注意……120
7.合計と平均……121
8.繰り返し……123
9.仕事のレベル差のコントロール……126

10.ルーチンを強制中断する……127
11.あらかじめ繰り返しの回数を指定しない方法……128
12.その他のルーチンの例……133

Step3 オペレーティングMUMPS

Chapter 7 データを削除する……136
1.データの削除……136
2.変数の親子関係……138
3.親亀こけると子亀もこける……139
4.データを削除するルーチン……141
5.データの削除による問題点……145
6.データが存在するかどうかを調べる……146
7.$DATA関数とは……148
8.条件によりデータリストを改造する……152
Chapter 8 データの兄弟をさがす……156
1.登録番号が飛び離れている場合の問題……156
2.空文字列の発見……157
3.次の添え字を捜す……159
4.繰り返しの方法……162
5.FOR命令とQUIT命令の組み合わせ……164
6.終わらないFOR命令……165
7.名前のリスト……167
8.DO命令の利用……172
9.平均年齢の計算……175
Chapter 9 データを並べ替える……177
1.文字列添え字……177
2.名前による索引グローバル……179
3.索引グローバルを利用した検索プログラム……183

4.文字列添え字の並ぶ順序……184
5.名前順のリスト……185
6.年齢順に並べ替える……187
7.添え字が複数あるときの$ORDER関数の働き……190
8.同一年齢の人を区別する……194
9.同姓同名のデータの並べ替え……197
Chapter10 もっと便利にルーチン改造……200
1.データ登録ルーチンの問題点……200
2.仕事の下請け……201
3.入力データを確認する……203
4.修正したい項目を指定する……205
5.登録済みのデータの修正……207
6.データの登録と修正，両用……209
7.メインドライバールーチン……211
8.職員名簿管理システムのまとめ……214
応用 MUMPSファイル
file 1 データベース……220
[1] 住所録……220
[2] 会費納入状況の記録……222
[3] 文献管理……223
[4] 顧客管理……224
file 2 文字列処理……226
[1] 文字列処理をする関数と演算子……226
[2] 日付の変換……229
[3] 入力データの検査……233
file 3 システムの連携……236
[1] 漢字処理……236

[2] 画面制御……238
[3] 図形をかく……241
[4] 他のシステムへデータを渡す……244
[5] ワープロと連携する……246
[6] 他のシステムの呼び出し……248
file 4 ユーティリティプログラム……251
[1] ルーチンのリスト……251
[2] %USユーティリティ……255
file 5 MUMPSの文法要約……257
[1] データ……257
[2] 変数……257
[3] 関数……258
[4] 演算子……259
[5] 式……262
[6] 命令……262
[7] ルーチン……264
[8] 間接指定……265
file 6 JIS（ASCII）コード表……266
INDEX……268
参考文献……271

SP-MUMPSは住友電工の登録商標です
一太郎　はジャストシステムの登録商標です
MS-DOS　はマイクロソフトの登録商標です
Simple Chartはステラシステムの登録商標です
EGR-98　はカノープス電子の登録商標です
VJE-α　はバックスの登録商標です
VJE-Σ　はアスキーの登録商標です

Step 0 ようこそMUMPSワールドへ

Step0ではMUMPSとは何か，その特徴は何か，そして何に使えるのかを概説します。

その上でこの本の構成と利用方法について説明します。

Chapter 0

プロフィルMUMPS

MUMPSとは

この章では，まず MUMPS（マンプスと呼んでください）とは何か，どのような特徴があるのか，何に使えるのかといった点について解説したあと「ビギニング MUMPS」の構成についてふれます。

1 MUMPSはデータベースシステム

世は情報社会です。情報を制することが覇を握る鍵です。

消費者の要求を的確につかみ，商品企画に反映させることができれば，より効率の良い投資ができるでしょう。ユーザーの過去の購買記録を管理しておけば，宣伝効果を上げることもできます。

いままでは個々のデータをカードやノートなどに記録しておいて，手作業でそれらを調べるしかありませんでした。ところがデータをコンピュータに蓄えておくと手作業ではとうていおよばない角度からの分析も可能となります。

この目的でデータを大量に蓄え，あとからそれをさまざまな用途に使うこと

fig 0-1　2次元の「表」で表わせるデータベース

番号	レコード名	演奏者	レコード会社
1	戯夜曼	岩崎宏美	ビクター
2	TWIN	早見 優	トーラスレコード
3	クロス・ポイント	カシオペア	アルファレコード
⋮	⋮	⋮	⋮

ができるように開発されたコンピュータシステムを，データベースシステムと呼びます。

この本で解説している MUMPS も，このデータベースシステムのひとつです。したがってデータを蓄えることであれば，MUMPS には無数の利用方法が考えられます。

2 MUMPS＝枝分かれ構造データベース

データをコンピュータにどのように整理して蓄えるかという技術は，大勢の人々が研究しておりさまざまの方式のものが考えられています。

そのなかで MUMPS のデータベースは，「階層型」といわれる種類に分類されます。

近年，パソコンの普及にともない分かりやすいデータベースシステムが必要とされています。リレーショナル・データベースといわれる種類のシステムがあり，これはデータベースを２次元「表」としてとらえることができるので素人にも理解しやすいといわれています。

ところが，現実のデータでは，２次元の「表」として考えるのが難しいことがしばしばあります。

fig 0-2　３次元の「表」

番号	レコード名	演奏者	曲　名
1	戯夜曼	岩崎宏美	恋孔雀 星遊劇 （その他）
2	TWIN	早見優	Wanna Change You ニュートラル （その他）
3	クロス・ポイント	カシオペア	スマイル・アゲイン スウェアー （その他）
：	：	：	：
10	サン・シティ	ボブ・ディラン ホール＆オーツ マイルス・デイビス （その他）	サン・シティ ノー・モア・アパルトヘイト レット・ミー・シー・ユア・I. D. （その他）
：	：	：	：

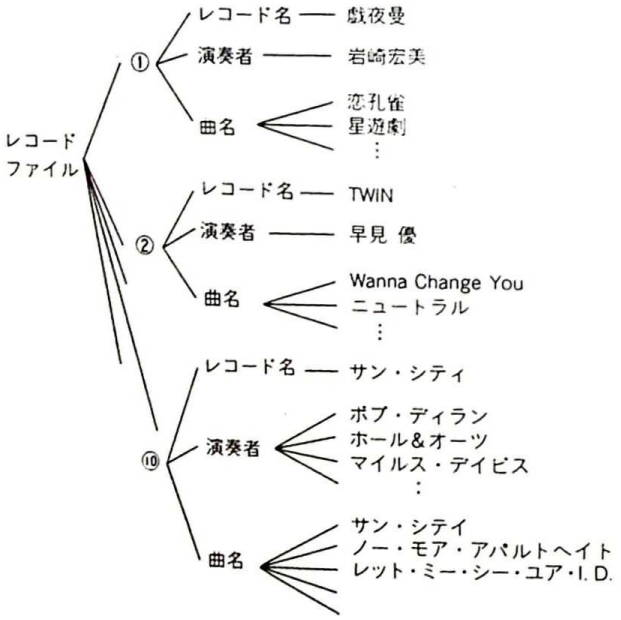
：１つの項目に複数のデータが入ると，３次元（以上）の表になってしまう。リレーショナル型データベースでは，このままでは処理できない

たとえばレコードを整理することを考えてください。１枚のレコードには複数の曲が収録されています。演奏者が２人以上入っていることもあります。このように単一の項目に複数のデータが存在する場合，これを単純な１枚の「表」に表すことはできません。データを「表」という構造だけで，それも効率よく管理するにはそれなりの知識と技術が必要となります。

MUMPS のデータベースは枝分かれ構造（階層構造）になっているので，ひとつの項目に複数のデータを入れることができます。曲目や演奏家が複数でも自然なデータ構造をそのまま反映させ，分かりやすく表現することができます。

さらに各データについてより細かい分類，たとえば各演奏家が何の楽器を使っているかといったデータをつけ加えることも，枝分かれを増やすだけですから簡単です。

fig 0-3　枝分かれ構造(階層構造)による表現



：MUMPSのデータベースは枝分かれ構造で，
ひとつの項目にいくつデータが入ってもよい

3 MUMPSは文章データを効率よく蓄える

以前，コンピュータは数値計算するものと思われていましたが，最近では文章や言葉のデータを処理することが多くなっています。ユーザー管理をする場合でも，名前や住所，会社名，役職など重要なデータの多くが言葉のデータです。

言葉のデータの問題は，個々のデータごとに文字数に大きなばらつきがあることです。名前の場合でも「森進一」は3文字，「薬師丸ひろ子」は6文字です。

住所に至っては大字・字，アパート名，棟番号，部屋番号などがついてやたらに長いものがあります。

このようにデータの長さは個々のケースで千差万別，最大の長さすら予測することが困難です。

ところがコンピュータシステムでは，コンピュータ側の負担を軽くするために，蓄えるデータの長さや数をあらかじめ決めてしまうことが一般的です。

このためデータ項目ごとに，十分な長さの記憶場所を確保しておかなくてはなりません。しかしあまり長く場所を取り過ぎると，用意した場所のうち使用せずに無駄になる部分も多くなります。

MUMPS のデータベースの特徴は，あらかじめデータの長さや項目数を宣言しておく必要がないことです。

データが発生し，入力された時点で，必要最小限の記憶場所が自動的に確保されます。このため記憶装置をきわめて効率よく使うことができ，個々のデータ量を予測するという無理な仕事から解放されます。

4 MUMPS=コンピュータ言語

MUMPS は，データベースシステムであると同時に，コンピュータプログラムを開発するためのコンピュータ言語でもあります。

コンピュータ言語としての MUMPS は，比較的 BASIC 言語に似た「手続き型」言語です。BASIC と同じようにコンピュータと会話をしながらプログラムを作ることができ，作ったプログラムをその場で実行することができるため，プログラムの開発・修正がきわめて容易です。

プログラミングの生産性は，COBOL の５から10倍，BASIC の２倍程度です。パソコン用リレーショナル型データベースシステムの dBASE-II などと同程度と考えられます。

BASIC，dBASE-II など他の言語に比べ，制限がきわめて少ないのも特徴です。

前述のとおり，データベースに関してあらかじめ長さや項目数を宣言しておく必要がありません。同時に扱えるデータファイルの数に制限はなく，一時的な記憶に利用する「変数」の数にも論理的な制限はありません。

また人工知能で有名な LISP などと同様，データとプログラムの区別がないのも特徴的です。この性質を利用して，プログラムを編集するプログラムや自己増殖するプログラムなども容易に作成できます。

用意されている文法要素が少ないのも特徴としてあげられます。たとえば命令は30足らずしかありません。

某社の BASIC 言語には90以上のコマンドやステートメントがあります。命令，関数，演算子の数が少なければ機能も低いような錯覚に陥りますが，ひとつ一つが極めて強力なので決して機能が劣ることはありません。

文法要素が少ないぶん，言語を覚えやすいという利点があります。

変数やラベルに使用が禁止される「予約語」がないので，文法を一部しか知らなくても，その範囲で自由にプログラムを書くことができます。

これらの特徴は初心者にとって非常にありがたいことです。

⑤MUMPSは標準化が進んでいる

パソコンの BASIC 言語がパソコンの機種ごとに異なるため，有用なプログラムを移植するのに多大な労力が払われています。このことが反省されたため最近流行のＣ言語などでは，標準言語を作る作業が進められているようです。

MUMPS は先人達の努力の結果，言語の標準化が他に類を見ないほど進んでいます。

1977年には，ANSI（米国国家規格協会――日本の JIS にあたる）が FORTRAN，COBOL についで３番目の標準言語として認可しました。さらに1984年には改訂版が標準として認められました。

この結果，プログラムやデータを別の MUMPS システムに移植するのがきわめて容易になっています。見知らぬアメリカ人が，見知らぬコンピュータで作成したプログラムを，そのまま国産のコンピュータで利用することができるのです。

6 MUMPSはさまざまのコンピュータで使用できる

MUMPS は1960年代末に，もともと DEC 社のミニコンピュータで開発されました。

その後その有用性が認められ，非常に多くのコンピュータに移植されまた新しく開発されています。

その結果，現在では AppleII のようなマイコンから，IBM 社の大型汎用コンピュータに至るまで，さまざまなコンピュータで利用することができるようになりました。

そして最近，国産のパソコン用にも MUMPS が発表され，日本でもこの強力なデータベースシステムを手軽に利用できるようになっています。

前述したとおり言語の標準化が進んでいるので，マイコンから汎用コンピュータまでまったく同じプログラムを走らせることができます。このことをうまく利用すると，はじめ小さなコンピュータを用いて，小さなリスクでシステムを開発し，軌道に乗ったところで大きなコンピュータに移植するということができます。

また MUMPS はもともとひとつのコンピュータを同時に複数の人が使いデータを共有するタイムシェアリング，データベースシェアリング・システムとして開発されてきました。したがって同時に複数処理をするのに必要な機能はすでに用意されています。

現在のパソコン用の MUMPS は，ハードウエアの制約から一時にひとつの仕事しかできません。

しかし将来パソコンの性能が向上し，同時に複数の仕事をさせるようになるかもしれません。その場合でも MUMPS の互換性・拡張性には問題がないでしょう。

7 MUMPSとネットワーク

コンピュータを用いたネットワークシステムや通信システムが脚光を浴びつつあります。

MUMPS を開発してきたミニコンピュータは，コンピュータ間の通信にも柔軟に対応できるという特徴を持っています。このため，MUMPS を用いたネットワークシステムも珍しくありません。京都大学付属病院や日本通運など大規模な MUMPS 利用施設の多くは，すでに複数のコンピュータを接続しています。

MUMPS を用いた「電子メール」システム（日本DEC）が発表されていますし，「電子掲示板」システムを MUMPS で作るのも容易だと思われます。

他のコンピュータ上のデータをあたかも自分のコンピュータ上にあるかのごとく扱うシステム（Data Tree／DT-MUMPS システム）もすでに開発されています。

8 MUMPSと人工知能

MUMPS のデータベース機能と文章データ処理機能をうまく利用すると，いわゆる「人工知能」システムを作ることができます。

Prolog システムのように，推論機能を持っているわけではありませんが，プログラムで実現することができます。

一方「知識ベース」といわれる，構造を持ったデータベースを構築することに適していると考えられています。

現にヨーロッパでは西独フランクフルト大学の MEDIUC システムなど MUMPS を利用したエキスパートシステムがいくつか開発されています。

MUMPS の機能を拡張して，リスト処理機能や Prolog 的な推論機構を組み込む試みも進んでいます（名古屋，マンプスシステム研究所 GNOSIS システム）。

9 MUMPSと数値計算

MUMPS は文章データの処理には向いていますが，数値計算に向いているとは思いません。多元連立方程式を解いたり，弾道の解析をするなら他の言語

を利用することを勧めます。

ただし数値計算ができないわけではなく，統計処理システムはいくつも発表されています（千葉大学医学部，CUPIDS システム）。大量の計算をするのでなければ十分実用的です。

標準 MUMPS には数値計算の関数がありませんが，パソコン用の MUMPS では sin，cos，exp，log のような関数も準備されています。

⑩MUMPSを何に使うか

MUMPS はデータベースとプログラム言語が結び付いた特異なシステムで，今まで述べたように，応用分野が非常に広範囲です。

何に用いるかは，結局あなたの創意工夫によります。柔軟な発想と粘り強さがあれば，思いもよらない利用方法があるでしょう。

MUMPS はもともと米国ボストン市にあるマサチューセッツ総合病院で開発された，汎用データベースシステムです。

MUMPS という名前は，Massachusetts General Hospital Utility Multi Programming System の頭文字を取ってつけられました。

病院で開発され，名前が mumps（流行性耳下腺炎・おたふくかぜ）ではありますが，このシステムは医療専用のものではありません。むしろ医療にも使える汎用性の高いデータベースシステムといったほうが正確に内容を示しているでしょう。

「ビギニングMUMPS」の構成

この本はデータベースシステムであり，プログラム言語である MUMPS システムの入門書です。

1 本書の目的

もともと，ほとんどコンピュータに触れたことがない人を対象にしていますが，BASIC 言語や，他のデータベースシステムを使ったことがある人にも読んで頂けるように配慮しました。

ただし，この本では分かりやすさを最優先にしており，MUMPS の機能のすべてを解説しているわけではありません。

MUMPS がもっともよく利用されるのはデータベースシステムとしての機能なので，それに焦点を当てています。他のシステムとの比較を必要とする人のために，本文で説明しきれない機能については「MUMPS ファイル」として別に解説しましたので参考にしてください。

さらに MUMPS の全機能を知りたい方は，巻末に参考文献をあげておきましたのでそれらを参照してください。

この本では解説の都合上，パソコンで作動する SP-MUMPS というシステムに焦点を当てています。そのため，SP-MUMPS に特徴的な機能もいくつか入っています。しかし多少使い方や名称が変わっても，他の MUMPS システムにも用意されている機能ばかりです。したがって，他のパソコンやミニコンの，MUMPS システムの学習用にもこの本は利用できると思います。

2 本書の構成

Chapter 0　MUMPS の特徴を簡単にまとめて解説しました。専門的な用語も出てくるので，初心者には少々難しいかもしれませんが，読み流してかまいません。

Chapter 1　MUMPS を使用する準備方法について解説しています

Chapter 2　実際に MUMPS を動かす方法をステップを踏み解説しています

Chapter 3　MUMPS のプログラムの作り方

Chapter 4　データの記憶方法

Chapter 5　変数の配列の利用方法

Chapter 6　データベースの作り方

Chapter 7～8　MUMPS の階層型データベースの操作機能

Chapter 9　データの並べ替えの方法

Chapter 10 データの登録プログラムを例にして，モジュール化したプログラムの開発

Chapter 5 から Chapter 10では以上の内容を「職員名簿」を例にとってデータを蓄えたり，蓄えたものを調べたりする方法を解説しています。

Chapter 10まで学習すれば，普通のデータベースの管理がひととおりできるようになるはずです。例にあげた「職員名簿管理」という仕事は，少し手直しすれば住所録や商品管理などに広く応用できます。

file 1 には別のデータベースの例をいくつかあげて，MUMPS のデータベース機能を紹介しています。

file 2 は MUMPS の文字列データ（言葉・文章のデータ）処理機能の概略を解説しています。ここを良く読めば，MUMPS の文字列処理の強力さが分かって頂けると思います。

file 3 では，MUMPS 以外のシステムとの連携の方法について解説しています。ここでは本文で解説に利用した SP-MUMPS システムを用いて，ワープロや作図システムとデータ交換したり，MUMPS からそれらを呼び出す方法について簡単に述べてあります。ただし，内容の性格上，SP-MUMPS に特有な機能を活用しています。多くの MUMPS に同じような機能が用意されていますが，あらゆる MUMPS システムに適用可能な内容ではありません。

file 4 では，便利なユーティリティプログラムをふたつ紹介しています。これも SP-MUMPS に特有な機能を利用していますが，少し手直しするとあらゆる MUMPS システムに適用できます。

file 5 は MUMPS の文法要素の要約です。他システムとの比較の参考になります。

file 6 は，ASCII および JIS コード表です。

巻末には MUMPS についてさらに知りたいかたのための参考文献をあげておきました。

③読み始める前に

MUMPS を習得するには，この本を読みながら実際にコンピュータで試すことをお勧めします。しばしばいわれることですが

「習うより慣れろ」

です。とくに，キーボードの苦手な人なら少しでもキーボードに慣れるようにしてください。タイプの速度は MUMPS の習得に要する時間に影響します。

またコンピュータの操作に入る前に，使用するコンピュータの電源の入れ方，フロッピーディスクの扱い方，キー操作の方法などを調べておいてください。多分コンピュータのメーカーが，初心者向けのテキストを用意しているはずです。

さらに，この本で例として扱う SP-MUMPS を操作するには，MS-DOS に関する知識があった方がよいので，MS-DOS の入門書か解説書を入手してください。最低限フロッピーディスクのフォーマットの方法と，コピーの方法を覚えてください。

最後にこの本の本文の解説に利用しているシステムは次のとおりです。

ハードウェア：PC9800シリーズ（日本電気製のパソコン）
ソフトウェア：MS-DOS（マイクロソフト社製のオペレーティングシステム）
SP-MUMPS（住友電気工業製の MUMPS システム）

Step 1 スターティングMUMPS

MUMPSはユーザーがデータを渡すと，何らかの演算をして，その結果を答えてくれます。

Step 1ではMUMPSにこのような仕事をさせるための基礎知識を説明します。

Chapter 1ではMUMPSを呼び出す手順を述べます。

Chapter2ではMUMPSと対話しながら作業を進めるための基本を述べます。

Chapter3では仕事の手順をMUMPSに教える方法を説明します。

Chapter 1

MUMPSを呼び出す

さあ MUMPS を使いましょう。

しかし，コンピュータの電源を入れるだけでは MUMPS を使うことはできません。この章では，MUMPS を準備する方法を解説します。

1,MUMPSを使うにはMS-DOSが必要

この本で解説している SP-MUMPS を用いるには

①パーソナルコンピュータ[1)]

② SP-MUMPS システム

③ MS-DOS システム[2)]

を用意しなくてはなりません。

MS-DOS は容易に入手できるので，マイコンショップなどに相談してください。またワードプロセッサや，表計算プログラムなどのプログラムに MS-DOS が組み込まれている場合があります。そのようなプログラムがあれば，その MS-DOS を流用できるはずです。

2,フロッピーディスクを準備する

マイコンショップから入手した「SP-MUMPS」の包みには，1 冊のマニュ

■notes

1) 1986年3月現在SP-MUMPSを利用できるパーソナルコンピュータは，日本電気PC-9801シリーズ，PC-98XA，N5200/05mk II，富士通FM16β，沖電気if800/60，IBM 5540，5550，5560です。この本は，PC-9800シリーズのコンピュータを想定して解説しています。

2) 「MS-DOS」にもいくつかの改訂版がありますが，バージョン2.0以降のものが必要です。IBMのパソコンではMS-DOSを，「日本語DOS」という名前で呼んでいます

アル（取り扱い説明書）と数枚のフロッピーディスクが入っています。

この中で，まず必要なのは，「SP-MUMPS システム」と「SP-MUMPS 基本データベース」という２枚のフロッピーディスクです。

そして「MS-DOS」のディスクも用意してください。

SP-MUMPS の包みには「基本データベース」のフロッピーディスクが１枚しか入っていません。フロッピーディスクの内容が壊れるのが心配な方はコピーを取っておいてください[3]（コピーの方法は MS-DOS の解説書参照）。

3，MUMPSを呼び出すための手順

SP-MUMPS を使うためには，次のふたつのステップを踏みます。

① MS-DOS を呼び出す

② SP-MUMPS を呼び出す

このふたつのステップについて，操作方法を説明します。MS-DOS の呼び出し方を知っている人は①を飛ばしてもかまいません。

1 MS-DOSを呼び出す

ここでは MS-DOS を呼び出す方法について解説します。

fig 1-1　用意するフロッピーディスク

notes

3）「基本データベース」は，少々手間はかかりますが，「インストレーション」という作業をすると作り直すことができるので，コピーしなくても問題ないでしょう。「インストレーション」をすると，データベースをハードディスク上におくこともできます。しかしこの本では，説明を簡略にするため，フロッピーディスクのみを扱うことを前提にします。詳細はSP-MUMPSの取り扱い説明書のうち「運用マニュアル」の第２章を読んでください

MS-DOS システムの入ったフロッピーディスクを用意してください。そして以下の手順に従ってください。

1．**コンピュータ本体の電源スイッチを入れる**

2．**フロッピーディスクを入れる**

MS-DOS システムのフロッピーディスクを 1 と書いてあるディスク装置に入れてください。

3．**リセットスイッチを押す**

1.と 2.の手順が素早く行われた場合はこの操作は省略してかまいません。

4．**日付を入力する**

数秒後，モニターテレビ画面に数行のタイトルが表示されます。そして画面の左端に

```
現在の日付は 1986-02-12 （水）です
日付を入力して下さい  :
```

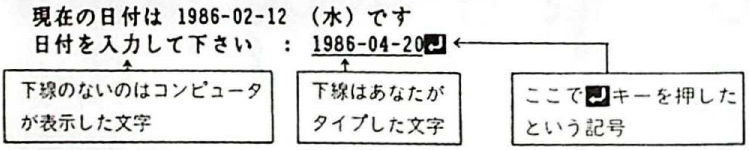
という文字が表示され，そのすぐ右側にカーソル（四角い記号）が点滅しているはずです。

もし，しばらく待っていてこれが表示されなければ，フロッピーディスクが正しく入っているかどうかを確かめて，2.から繰り返してください。

上記のメッセージが表示されたら「現在の日付」を確かめてください。

もし間違いなければ↵キー（リターンキー）を打ってください。

間違っていれば，19yy-mm-dd という形式で日付をタイプして，最後に↵キーを打ってください。ここで yy は西暦年，mm は月，dd は日です。たとえば1986年4月20日なら 1986-04-20 とタイプしてください。



5．**時刻を入力する**

次に

```
現在の時刻は 16:30:13.00  です
時刻を入力して下さい  :
```

というメッセージが表示されるはずです。これが表示されたら「現在の時刻」を確かめてください。

もし間違いなければ↵キーを打ってください。

もし間違っていれば hh:mm:ss という形で時刻をタイプして，最後に↵キーを打ってください。ここで hh は 24 時間制の時間，mm は分，ss は秒です。秒は省略できます。午後5時12 分なら 17:12です。

6．MS-DOS が起動されたことを確かめる

日付，時刻の入力が終わると，画面の左端に

```
A>
```

という表示が出て，そのすぐ右側にカーソルが点滅しているはずです。

これで MS-DOS が起動され（呼び出され）ました。

注意 フロッピーディスクを入れた状態で，コンピュータ本体の電源を入れたり切ったりするとフロッピーディスクの内容が壊れることがあります。電源スイッチを使うときには，必ずフロッピーを取り出しておいてください

2 MUMPSを呼び出す

MS-DOS が起動された状態で，次に SP-MUMPS を呼び出します。

1．フロッピーディスクをセットする

MS-DOS のフロッピーディスクを取り出します。

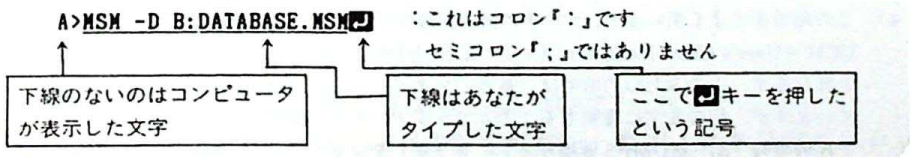
「SP-MUMPS システム」のフロッピーディスクを '1' のディスク装置に，「基本データベース」のフロッピーディスクを '2' のディスク装置に入れます。

2．MUMPS を呼び出す

上記の状態で

```
MSM -D B:DATABASE.MSM
```

とタイプして，最後に↵キーを打ってください。



ここでタイプしたときに，モニターテレビに 'モトモ ホシコ' などとカタカナで表示されてしまったら，カナロックキーを一度押してからタイプしなおしてください（キーボード右上のBSキーを押すと文字が消えます)。

また，もし 'msm -d b:database.msm' と小文字で表示された場合は，ここでは問題はありませんが，下記の 4．のところで問題が生じるのでCAPS ロック・キーを一度押しておいてください。

3．コンフィギュレーションの選択

数秒すると，いろいろメッセージが現れてから

```
スタート時のコンフィギュレーションを入力して下さい <DFLT>
```

というメッセージが表示されます。

ここでは意味は分からなくて結構です。↵キーのみを打ってください。

4．暗号を入力する

さらにまた 2-3秒すると

```
SP-MUMPS .... Line #1  UCI:
```

というメッセージが現れます。ここで決められた暗号を入力してください（差し当たり MGR:XXX という暗号を入力することができます)。

```
SP-MUMPS .... Line #1  UCI:MGR:XXX↵
```

この MGR:XXX は一種の暗号で，この暗号を知らないと MUMPS が使えないようになっています。[4)]

5．MUMPS が起動されたことを確認する

モニターテレビ画面左端に

```
>
```

という記号（プロンプト）が表示され，そのすぐ右側にカーソルが点滅していれば MUMPS の起動に成功した証拠です。

■notes

4) この暗号をうまく用いると，データの機密保護ができます。この暗号のうち，MGRをUCI(＝User's Class Identifier)といい，XXXをPAC(＝Programmer's Access Code)と呼びます。このふたつの暗号は，基本データベースには，MGR：XXX と登録されていますが，他の文字に変更することもできます。その方法はSP-MUMPSのマニュアルのうち「SP-MUMPS運用ガイド　第3章」を参照してください。

MUMPSの起動の方法

- 電源を入れる
- MS-DOSのフロッピーディスクをフロッピーディスク装置の '1' に入れる
- リセット・ボタンを押す
- 日付と時刻を入力する
- A> という表示が出たら，MUMPSシステムのディスクをディスク装置 '1' に，基本データベースのディスクをディスク装置 '2' に入れる
- MSM -D B:DATABASE.MSM↵とタイプする
- スタート時のコンフィギュレーションを入力してください <DFLT> というメッセージには↵キーのみを打つ
- SP-MUMPS.... Line #1 UCI:
  というメッセージが表示されたら決められた暗号をタイプする(差し当たりMGR:XXXという暗号が使用できる)

4,MUMPSの仕事を終了する　HALT命令

運転の方法を学ぶ前にエンジンの停止方法を先に解説します。

MUMPS を停止する方法は簡単です。H 1 文字をタイプします。

```
>H↵              : このHをHALT命令と呼びます
```

すると，1-2 秒後に次のように表示されます。

```
Exit
A>               : MS-DOSが起動された状態になりました
```

場合によってこのようにならないことがあります
「MUMPS終了時に生じるエラーについて」を参照してください

この状態でフロッピーディスクを取り出して電源を切ってください。これで終わりです。

◎MUMPS終了時に生じるエラーについて

MUMPS を終了するときに，MUMPS を呼び出したディスク装置にフロッ

ビーディスクが入っていないと

```
ドライブの準備ができていません <ドライブ A:> <読み取り中>
中止<A>，もう一度<R>，無視<I>?
```

というメッセージが出ます。このときは，MS-DOS のフロッピーディスクを入れて，R をタイプしてください。

さらにフロッピーディスクが入っていても，そのディスクに COMMAND.COM という MS-DOS のプログラムファイルが入っていないと

```
正しいディスクをカレントドライブに差し込み，
どれかキーを押して下さい
```

というメッセージがでます。この場合は，MS-DOS のフロッピーディスクに差し替えて，何でも良いですからどれかキーを押してください。すると

```
A>
```

のプロンプトが表示されます。これで電源を切ることができます。

「正しいディスクをカレントドライブに……」というエラーを生じないようにするには，MUMPS を起動したディスク装置に MS-DOS の COMMAND.COM というファイルの入ったディスクを入れておいてください。

SP-MUMPS システムのフロッピーディスクには MS-DOS が入っていないので，COMMAND.COM をこれにコピーしておくか，MUMPS 起動後は，MS-DOS のディスクに差し替えておくとよいでしょう。

- MUMPSを終了するにはHALT命令を用いる
- A> と表示されてからフロッピーディスクを取り出して電源を切る
- フロッピーディスクを入れたまま電源を切ったり，HALTせずにフロッピーディスクを取り出すとフロッピーディスクが壊れる可能性がある

Chapter 2

MUMPSと対話する

この章ではまず MUMPS との対話の方法について学びます。前章で解説したとおりの方法で MUMPS を呼び出してください。

1, デタラメにキーをタイプしてみる

MUMPS が呼び出されていると，モニターテレビの画面の左端に

>

という記号が表示され，そのすぐ右側にカーソルが点滅しているはずです。この「>」記号は MUMPS が，あなたのメッセージを受け付ける準備が整っていることを示しています。

もしこの記号が出ていなければ↵キーを打ってください。何かメッセージが表示されたあとでこの状態になるはずです。それでもだめな場合は，うまく MUMPS が起動されていないのかもしれません。操作の分かる人にたずねてください。

では，デタラメにキーをたたいてみましょう。普通の力でたたいている限りはコンピュータは壊れる心配はありません。たたいたキーに対応する文字がモニターテレビ画面上に現れるはずです。

```
>YLPIOWUERAF,ZNV.                    ：デタラメにタイプしてみました
```

最後に↵キーを打ってください。

注意

この本では，分かりやすくするため，例を示すときユーザーがタイプした文字には下線を引き，MUMPSが表示したメッセージには下線を引かないことにします。実際のモニターテレビ画面にはこの下線は表示されません

また↵キー（リターンキー）を打っても何も文字は表示されませんが，このキーを打つということを明確に表すために，↵という記号を入れておきます

```
>YLPIOWUERAF,ZNV.↵                    :↵キーを打ちました

<CMMND>  定義されていないコマンドが使われました   } :MUMPSが
 0) *XECUTE*:: YLPIOWUERAF,ZNV.                    } 返してきた
<CMMND>*XECUTE*:::1:0 YLPIOWUERAF,ZNV.             } メッセージです
```

この例では「〈CMMND〉 定義されていないコマンドが使われました」というメッセージが MUMPS から返されました。これは MUMPS があなたの言っていることを理解することができないといっているもので，エラーメッセージと呼ばれます。

よく現れるエラーメッセージにはこの他「〈SYNTX〉 シンタックスが間違っています」などがあります。エラーメッセージの内容を理解し慣れることは，MUMPS の習得に大きく役立ちますが，今は細かいことを気にしなくても結構です。

- 「>」という記号が画面の左端に表示されていたらMUMPSの命令をタイプすることができる
- 画面にはカーソルが点滅しておりタイプした文字はそのカーソルのあった場所へ表示される
- ユーザーはメッセージの最後に，↵キーを打たなくてはならない
- MUMPSはユーザーからのメッセージが理解できないと，エラーメッセージを表示する
- ↵記号はリターンキーを打ったことを示す
- この本の例のうち，下線の引いてある文字はユーザーがタイプした文字であることを示す

2,簡単な計算をさせよう

WRITE命令

MUMPS に理解できるメッセージをタイプしてみましょう。次の通りにタイプして

```
>W 3+5
```

下線はあなたがタイプした文字という意味です

：Wの後に空白文字（以降，"空白"と呼びます）を正確に1個だけ入れてください。空白はキーボードの一番手前にある長いキー(スペースバー)を打ってください

最後に↵キーを打ってください。

```
>W 3+5↵                    :Wや↵キーの前には空白を入れないこと
8
>
```

となるはずです。もしエラーメッセージが表示されたらどこかにタイプミスがあるはずです。よく見てください。Wの後に1個だけ空白が入っていますか。その他に余分な空白は入っていませんか？

さて，Wは WRITE 命令といって，それに続く3＋5という式を計算してその結果を表示することを MUMPS に要求するものです。

MUMPS は電卓とは異なり3＋5という式だけをタイプしても計算してくれません。計算した結果をどうするのかを指示しなくてはなりません。ここでは WRITE 命令で結果を表示することを指示しているのです。

いろいろな計算をさせてみましょう。

```
>W 3*5↵
>W 3-5↵
>W 3/5↵
>W 1+2+3+4+5↵
>W 1+2*3↵
```

どのような結果が表示されましたか？

エラーメッセージが表示されたら，どこかにタイプミスがあったはずです。もう一度正確にタイプしなおしてください。

①Wの後に空白を1個入れること

②余分な空白を入れないこと

③最後に↵キーを打つこと

を忘れないようにしてください。

ところで3＊5は3×5のことです。掛け算を表す記号『×』はアルファベットのXとまぎらわしいので，MUMPS では『＊』という記号（アステリスクと呼びます）を用います。

一方，割り算は『/』記号を用います。3/5は3を5で割るということです。

もうひとつ注意することがあります。W1＋2＊3の場合です。普通の計算では1＋2×3の計算結果は7ですが，MUMPS では9になります。これは

MUMPS の加減乗除の計算に優先順位がないからです。＋，－，＊，/はすべて左端から順に計算します。

この順序を変更する必要がある場合にはカッコ（　）を用いてください。たとえば

```
>W 1+(2*3)↵
7
>
```

となります。

いろいろな計算をさせてみましょう。

計算規則を理解するためにはいろいろな場合を試してみるのが一番です。

```
>W 3*3+3/4↵
3
>W 3*(3+3)/4↵
4.5
>W 3*3+(3/4)↵
9.75
>W 3*(3+3/4)↵
4.5
>W 100/4/5↵
5
>W 100/(4*5)↵
5
>W 100/4*5↵
125
>W 1/3↵
.33333333333333333          5)参照
>
```

■notes

5) SP-MUMPSの数値計算の有効数字は17桁あります。
標準MUMPSでは，プログラムの互換性を保証するため，数値計算の精度は有効数字9桁以内と決められています。
ユーザーは使用中のMUMPSから他のMUMPSへプログラムを移植するためには，有効数字9桁で考えておく必要があります

回間違ってタイプしたときの修正方法

①↵キーを打つ前にタイプミスに気が付いたときは，キーボードにBSと←というキーがあるのでいずれかを押してください。最後に入力した文字から消えて行くはずです。問題の文字まで消えたらもう一度そこからタイプしてください。

②↵キーを打ってしまってからではタイプミスは修正できません。多分すぐにエラーメッセージが表示されるはずです。もう一度始めからタイプしてください。

- WRITE 命令は，それに続く式を計算してその計算結果を表示する
- WRITE 命令は「W」1 文字で表される
- 「W」と計算式の間には空白を 1 個入れる
- 計算式と↵キーの間には空白を入れてはいけない
- ＋，－，＊，／はそれぞれ加算，減算，乗算，除算を表す
- 加減乗除には優先順位がなく，左端から順に計算する
- 計算順序を変更するにはカッコ（ ）を用いる
- タイプミスは↵キーを打つ前なら，BSまたは←キーを用いて修正できる

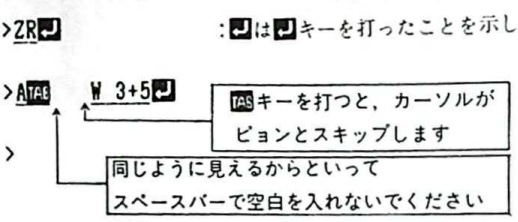
3，計算手順を教え込む ——ルーチンを作る

今までの命令の仕方では MUMPS は計算結果や手順をすぐに忘れてしまいます。同じ計算をさせるには，同じ命令を再度タイプしなければなりません。これが 3 ＋ 5 のように簡単な計算であればよいのですが，もっと複雑な式になると，何回も同じ式をタイプするのは面倒になります。

MUMPS には，ユーザーの必要とする仕事の手順をあらかじめ教え込んでおいて，後から何回もその手順を繰り返すことのできる方法があります。

この教え込んである手順のことを「ルーチン」と呼んでいます[6]。ここではこのルーチンの作り方を解説します。

まず次のようにタイプしてください。



ZR は ZREMOVE 命令といいます。これの意味は後述します。

ここでは一種の「おまじない」と思ってください。ZR の次に WRITE 命令をタイプしましたが，MUMPS は，3＋5の結果を出力しませんでした。それは，この行の内に TAB があるからです。ユーザーがタイプしたメッセージの中に TAB が含まれていると，MUMPS はそれをルーチンの一行とみなし，すぐに実行せず記憶するだけです。

次に続けて

```
>B TAB W 3*5↵
```

とタイプします。やはりこれも記憶されるのみで実行はされません。もしエラーが出たら ZR からやりなおしてください。

これで2行のルーチンを MUMPS に教え込みました。

注 意

Aと TAB とWの間には空白を入れてはいけません。また，TAB を打つとカーソルが右の方へビョンと飛びますが，これと同じようにみえるからといって TAB の代わりに空白を入れてはいけません。TAB と空白はMUMPSにとってはまったく違うものなのです。

- MUMPSにあらかじめ教えておく計算手順のことをルーチンという
- ユーザーのメッセージに TAB キーが入っていると，MUMPS はそのメッセージをルーチンの1行とみなし，すぐには実行しない
- TAB キーを押すと，カーソルが何文字分かジャンプするが，これは空白文字とは異なる

notes

6) コンピュータに教え込んである計算手順のことを，一般には「プログラム」と呼びます。MUMPSではこのプログラムの物理的な最小単位を「ルーチン」と呼び，ルーチンがいくつか集まってひとまとまりの仕事をするプログラムを構成します

4,ルーチンの内容を調べる

今入力したルーチンの内容を調べてみましょう。

```
>ZP↵
```

とタイプしてください。

```
>ZP↵
A       W 3+5
B       W 3*5


>
```

となるはずです。ZP とは ZPRINT 命令といって，現在記憶しているルーチンの内容を表示する命令です。したがって，この場合前にタイプした2行のルーチンが表示されました。

> • ZP はZPRINT 命令といい，そのとき記憶しているルーチンの内容を表示する

5,ルーチンを実行する

ここで

```
>D A↵
```

としてください。これは DO 命令といって，Aという行からそのルーチンの最後まで実行（DO）しなさいという命令です。ここで，DO 命令が正しくタイプされているにもかかわらず，もしエラーが出るようなら ZR からやりなおしてください。

```
>D A↵      ：Dや↵キーの前に空白を入れてはいけません。
815          DとAの間には空白を1個入れてください
>
```

MUMPS は815と表示しました。これはまず行AのW　3+5を実行して8を，行BのW　3＊5を実行して15を表示してきたのです。何回繰り返しても同じ動作をします。なぜなら，MUMPS は，前にタイプしたルーチンをずっと

記憶していて，それを繰り返し実行しているからです。

```
>D A↵
815
>D A↵
815
>
```

• ルーチンを実行するにはDO命令を用いる

6,もう一度繰り返してみよう

復習しましょう。

```
>ZR↵                          ：おまじない，これを忘れないこと

>A TAB  W 3+5↵                ：ルーチンの入力

>B TAB  W 3*5↵                ：ルーチンの入力
                                余分な空白を入れないように注意
>ZP↵
A       W 3+5                 ：ルーチンの内容を表示します
B       W 3*5


>D A↵                         ：ルーチンの実行
815
>D A↵                         ：ルーチンの実行
815
>
```

- MUMPSに教え込んでおく計算手順のことをルーチンという
- 新しくルーチンを作るときにはまずZRとタイプする
- ルーチンを入力するときは行の名前とその後にTABを打ち，そして命令をタイプする
- ZPは入力したルーチンを画面に表示する
- D Aはルーチンの行Aから実行を始める

7,表示を見やすくする

もう少しいろいろな計算をしてみましょう。次のようにタイプしてください。

```
>ZR↵                    ：おまじない，おまじない

>A1TAB  W 3+5↵

>A2TAB  W 3-5↵          ：TABの前後には空白を入れてはいけません

>A3TAB  W 3*5↵

>A4TAB  W 3/5↵
```

うまくタイプできましたか？　ZPRINT 命令で確かめてください。

```
>ZP↵
A1      W 3+5
A2      W 3-5
A3      W 3*5
A4      W 3/5
```

ではこのルーチンを実行します。どうなりますか？

```
>D A1↵
8-215.6
>
```

何が起こったのでしょうか？

4つの計算結果が1行につながって表示されました。これではどれがどの結果か分かりません。計算結果の間に空白を入れてみましょう。

```
>ZR↵
>B1TAB  W 3+5↵
>B2TAB  W " "↵      :『"』と『"』の間に空白を１個入れます
>B3TAB  W 3-5↵
>B4TAB  W " "↵
>B5TAB  W 3*5↵
>B6TAB  W " "↵
>B7TAB  W 3/5↵
>ZP↵                :ZPRINT命令でこのルーチンを表示します
B1      W 3+5
B2      W " "
B3      W 3-5
B4      W " "
B5      W 3*5
B6      W " "
B7      W 3/5
>D B1↵
8 -2 15 .6
>
```

分かりやすくなりました。

ここでまた新しい構文が現れました。すなわち

```
W " "
```

です。これは，空白を１個表示するという意味です。

引用符『"』ではさまれた文字はそのまま表示されます。例をあげます。いろいろな例を考えて確かめてください。

```
>W "ABC"↵
ABC
>W "3+5"↵
3+5
>W 3+5↵          :"(引用符) がないと
8
>
```

この例でW　3+5とW　"3+5"の意味が異なることに十分注意してください。

W　3+5は3+5の計算結果を表示しますが，W　"3+5"は3+5という引用符ではさまれた文字をそのまま表示します。

- WRITE命令は計算結果を続けて表示し，データの間に自動的には空白を入れない
- 『"』記号ではさまれた文字は，その文字そのものを表す

8,縦に並べて表示する

今度は計算結果を縦に並べてみます。次の通りにタイプしてください。

```
>ZR↵

>C1TAB  W 3+5↵

>C2TAB  W !↵

>C3TAB  W 3-5↵

>C4TAB  W !↵

>C5TAB  W 3*5↵

>C6TAB  W !↵

>C7TAB  W 3/5↵

>ZP↵
C1      W 3+5
C2      W !
C3      W 3-5
C4      W !
C5      W 3*5
C6      W !
C7      W 3/5

>D C1↵
```

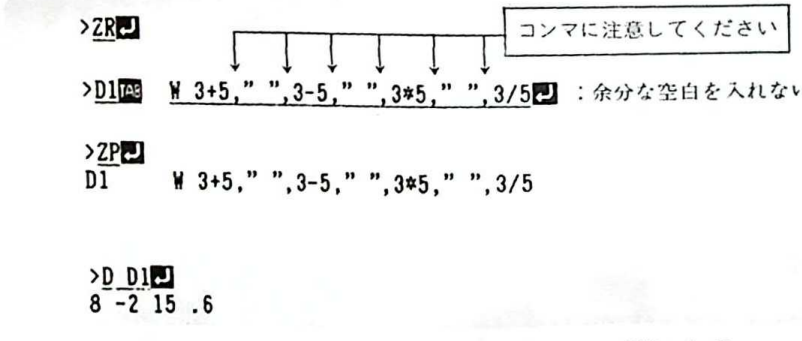
```
8
-2                    ：計算結果が縦に並んで表示されました
15
.6
>
```

前のルーチンでは計算結果が1行に表示されましたが，W ” をW！に替えると計算結果が縦に並びます。W！があると，次に表示されるデータは，画面の次の行の左端に表示されます。このように表示位置を次の行の左端に移すことを「復帰改行」すると言います。

- W！という命令があると，その次のデータは次の行の左端に表示される
- 表示する位置を次の行の左端に移動することを「復帰改行」という

9,WRITE命令を一行で書く

今までのルーチンは WRITE 命令を何行にも分けてきました。これらをすべて1行にしてしまうこともできます。この方がタイプする文字数も行数も少ないので一般的です。



これは結果を1行に表示する例です。3+5や ” ” の間にあるコンマ『,』に注意して正確にタイプしてください。

コンマがなかったり多過ぎたり，余分な空白が入っているとエラーになりま

す。

次は結果を縦に並べて表示する例です。

```
>ZR↵

>E1TAB  W 3+5,!,3-5,!,3*5,!,3/5↵

>ZP↵
E1      W 3+5,!,3-5,!,3*5,!,3/5


>D E1↵
8
-2
15
.6
>
```

ひとつの WRITE 命令の中に，表示する内容や復帰改行の『！』記号をいくつも入れるときはそれらをコンマで区切ることを覚えてください。これら表示すべき内容や『！』を WRITE 命令の「引数」（ひきすう，またはいんすう）と呼んでいます。したがって「引数と引数の間にはコンマを1個ずつ入れる」と表現できます。また WRITE，DO 命令のWやDを「命令語」といい，命令語と引数の間には空白を1個入れます。

この一連の例の最後として，次のルーチンを試してください。

```
>ZR↵                                  ：この「おまじない」を忘れないこと!!

>F1TAB  W !,"3+5=",3+5↵               ：コンマを忘れたり、余分な空白を入れない
                                        こと!!
>F2TAB  W !,"3-5=",3-5↵

>F3TAB  W !,"3*5=",3*5↵

>F4TAB  W !,"3/5=",3/5↵

>ZP↵
F1      W !,"3+5=",3+5↵
F2      W !,"3-5=",3-5↵
F3      W !,"3*5=",3*5↵
F4      W !,"3/5=",3/5↵
```

```
>D F1↵
```

空白1個

```
3+5=8

3-5=-2
3*5=15
3/5=.6
>
```

何が起こったのかよく考えてください。『 』ではさまれた文字の並び"3+5"と，式3＋5の違いを十分理解してください。

- W "ABC"は，ABCという3文字がそのまま表示される
- WRITE命令のWやDO命令のDのように命令の種類を表す文字を命令語という
- W 3+5,W" ABC",W !などのWRITE命令の3+5,"ABC",!のことをこれらのWRITE命令の引数という
- 命令語と引数の間には空白を1個入れる
- 引数はコンマで区切って続けることができる
- W!とすると以後に表示されるデータは画面の次の行の左端に表示される

10,ルーチンを修正する

タイプミスなしにルーチンを入力することができましたか？　これまでタイプミスしたときははじめから入力しなおすことしか解説していませんでした。しかしこれでは長いルーチンを入力するには非常に不便なことです。

MUMPS にはルーチンを1行ごとに入れ替える機能があります。

いま次のようにルーチンを入力してしまったとします。

```
>ZP↵
F1    W !,"3+5=",3+5
F2    W !"3-5=",3-5      ：この行がおかしい！
F3    W !,"3*5=",3*5
F4    W !,"3/5=",3/5

>
```

行 F2 にコンマが1つ足りません。これを修正するために，次のように入力してください。

```
>ZR F2↵

>F2TAB  W !,"3-5=",3-5↵
```

確認しましょう。

```
>ZP↵
F1      W !,"3+5=",3+5
F2      W !,"3-5=",3-5      ：コンマに注目
F3      W !,"3*5=",3*5
F4      W !,"3/5=",3/5
```

行 F2 が修正されているのが分かりますか。ここで

```
>ZR F2↵
```

の ZR は例の「おまじない」の一種で，ZREMOVE 命令といい，ZR F2 は行 F2 を削除するという意味です。削除した直後に新たに行を入力すると，削除された所へ自動的にその新しい行が挿入されます。

ここで十分注意が必要なのは修正すべき行を削除したら，**その直後に**正しい行を入力しなくてはならないことです。行を削除したあと何か操作をしてから新しい行を入力すると，それはどこへ挿入されるかは一概には決められません。どこに挿入するかを指定する方法は次章で述べます。

行単位で修正することは上記のような方法で可能ですが，1行が長くなるとこれも大変なことです。したがって MUMPS には多くの場合，文字単位で修正できる機能（エディタ）があります。これについても次章で述べます。

- ルーチンの入力ミスをしたときは，ZREMOVE 命令で指定した行を消し，その直後に正しい行を入力することで修正できる

Chapter 3

ルーチンについて

コンピュータにはさまざまな作業の手順を覚えさせることができます。覚えさせたひとまとまりの手順のことを MUMPS では「ルーチン」と呼んでいます。

この章では MUMPS のルーチンの構造，作り方，修正の仕方などを解説します。細かいことを覚えるのは少々苦痛ですが，これが基本なのでしばらく我慢してください。

1, ルーチンの構造

MUMPS のルーチンはいくつかの「行」からなっています。そして各行は，

行ラベル TAB 命令　　または　　TAB 命令

という形をしています。「行ラベル」は行を識別するための名前です。付けなくてもかまいません。リスト 3-1 にルーチンの例を示します。

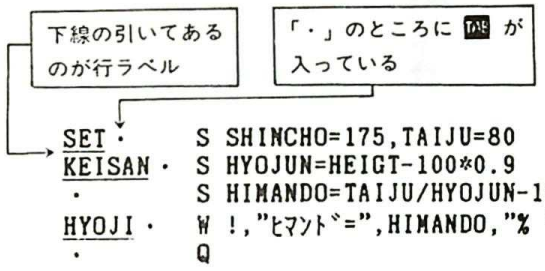
list 3-1

```
SET     S SHINCHO=175,TAIJU=80
KEISAN  S HYOJUN=HEIGT-100*0.9
        S HIMANDO=TAIJU/HYOJUN-1*100
HYOJI   W !,"ﾋﾏﾝﾄﾞ=",HIMANDO,"% "
        Q
```

このルーチンは 5 つの行からなっています。

SET, KEISAN, HYOJI という 3 つの行ラベルがあり，行 KEISAN と行 HYOJI の間と，最後の行には行ラベルがついていません。

TAB は実際には何も表示されませんが，各行の命令の先頭がそろっていることから TAB が入っていることがわかります（ TAB には次の文字の先頭をそろえる働きがあります）。



行ラベルはルーチン内の各行に付ける名前なので，同一ルーチン内で同じ名前は許されません。行ラベルには

①アルファベットの大文字

②数字

が使用できます[7]。ただし

③先頭が数字の場合にはすべて数字

でなくてはなりません。いずれにしても

④最大長８文字まで[8]

です。

行ラベルは行の前後関係とは無関係です。ルーチンを作るとき，入力した順番に行が並べられます。したがって，行9999の後に8888, 7777があってもかまいません。

次に行ラベルの例をあげます。

```
X  AZ  ABC  A123  R2D2  C3P0  ABCDEFGH  Q2345678
0  1  5  12  999  8765432  00  000  010
```

数字ばかりの行ラベルのうち，０と00や１と01などは異なった行ラベルとし

■notes

7) 先頭の文字に限り，「%」記号を用いることもできます。また，SP-MUMPSでは，カナや漢字も使用できます。しかしカナや漢字は世界的な標準仕様ではありません。作ったルーチンを他のMUMPSに移すことができなくなるので，使用しないほうがよいでしょう

8) 標準MUMPSの規定では，行ラベルやルーチン名(Chapter3-9参照)，変数名(Chapter4参照)は最大15文字まで使用できますが，MUMPSが認識するのは8文字までとなっています。このため最大８文字と覚えておいた方がよいでしょう。

てみなされます。次のような行ラベルは許されません。

```
$CHAR  #1  3.14159  &H0A  %%  100%  ABC%  160YEN
```

- ルーチンはいくつかの行からなっている
- 各行は行ラベルと命令からなりその間に TAB が入っている
- 行ラベルは付けても付けなくてもよい
- ひとつのルーチンに同一の行ラベルを使用してはいけない
- 行ラベルはすべて数字か，第1文字がアルファベットの大文字で，第2文字以降がアルファベットの大文字または数字である

2,ルーチンの作りかた

ルーチンを作る方法は，第1章で解説しましたが，ここでもう一度復習しましょう。

```
>
```

の記号が表示されている状態で

行ラベル TAB 命令 ↵　　または　　TAB 命令 ↵

と入力します。たとえば

list 3-2

```
F1      W !,"3+5=",3+5
        W !,"3-5=",3-5
F3      W !,"3*5=",3*5
        W !,"3/5=",3/5
```

という4行のルーチンを入力するには

```
>ZR↵                          :ZREMOVE命令（おまじない）
>F1TAB  W !,"3+5=",3+5↵       :行F1の入力
>TAB    W !,"3-5=",3-5↵       :行ラベルのない行の入力
>F3TAB  W !,"3*5=",3*5↵
>TAB    W !,"3/5=",3/5↵
```

とします。この場合の最初の ZREMOVE 命令は後述します。

前述の通り，行は原則として入力順に並べられ，行ラベルの論理的な順番とは無関係です。 F3 の後に F2 という行ラベルの行を入力すると，行 F3 の方が先になります。

- ルーチンの入力には行ラベルと命令の間に TAB キーを入れる
- ルーチンの行は入力順に登録される

3,ルーチンの内容を表示する　ZPRINT命令

入力したルーチンを画面に表示するには ZPRINT 命令を用います。
何も言わずに 'ZP' とすると，ルーチン全体が表示されます。

```
>ZP↵
F1      W !,"3+5=",3+5      ：行ラベルはF1
        W !,"3-5=",3-5
F3      W !,"3*5=",3*5
        W !,"3/5=",3/5

>
```

行ラベルを指定するとその行が表示されます。

```
>ZP F1↵
F1      W !,"3+5=",3+5

>
```

行ラベルのない行は，ラベルのある行から何行目かを指定します。

```
>ZP F1+1↵
        W !,"3-5=",3-5      ：行F1の次の行が表示されました

>ZP F3+1↵
        W !,"3/5=",3/5

>ZP F1+2↵
F3      W !,"3*5=",3*5      ：行F3は行F1の2行後です

>
```

先頭から何行目かを指定することもできます。

```
>ZP +1↵
F1      W !,"3+5=",3+5          ：ルーチンの先頭の行です

>ZP +4↵
        W !,"3/5=",3/5          ：第4行目です

>
```

コロン『：』を用いると指定した行以降をすべて表示します。

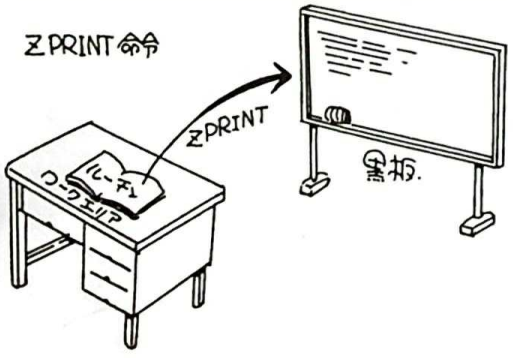
```
>ZP +2:↵                        ：コロン「：」に注意
        W !,"3-5=",3-5            セミコロン「；」ではありません
F3      W !,"3*5=",3*5
        W !,"3/5=",3/5

>
```

いろいろな場合を確かめてください。存在しない行を指定すると

〈LINER〉 定義されていない行ラベルが参照されました

というエラーメッセージが表示されます。

- ZPRINT命令はルーチンの内容を表示するのに用いる
- ZP　　はルーチン全体を表示する
- ZP行指定　は指定された行のみを表示する
- ZP行指定：　は指定した行以降の行をすべて表示する
- ZP行指定1:行指定2　は指定範囲の行を表示する
- 行指定の形式は，行ラベル，行ラベル＋整数，＋整数の3つがあり，それぞれその行ラベルの行，行ラベルの行の次から整数番目の行，先頭から整数番目の行という意味である



4,行を削除する

ZREMOVE命令

ZREMOVE 命令はルーチンの一部，または全部を削除します。この命令にも ZPRINT 命令と同じような3種類の形があります。確かめてみましょう。

```
>ZP↵                              ：ルーチン全体の表示
F1      W !,"3+5=",3+5
        W !,"3-5=",3-5
F3      W !,"3*5=",3*5
        W !,"3/5=",3/5

>ZR F1+1↵                         ：行F1の次の行を削除します

>ZP↵                              ：確かめましょう
F1      W !,"3+5=",3+5
F3      W !,"3*5=",3*5            ：行F1の次の行がありません
        W !,"3/5=",3/5

>ZR F3:F3+1↵                      ：さらに行F3から行F3+1を削除します
                                    コロンに注意
>ZP↵
F1      W !,"3+5=",3+5            ：行で残ったのは行F1だけ

>ZR↵                              ：全行を削除します

>ZP↵

                                  ：何も表示されません
>
```

この命令についてもさまざまな場合を試してください。

これまで 'ZR' という命令をルーチンを入力する前に「おまじない」として使ってきましたが，実はこれはルーチン全体を消す ZREMOVE 命令だったのです。

この命令は，そのとき MUMPS がコンピュータ内部のワークエリア（仕事場所）と呼ばれる記憶場所に記憶しているルーチン全体を消してしまいます。したがって新しいルーチンを作るときにはまずこの命令を用います。そうしな

いと古いルーチンがコンピュータ内部に残っているので，それが混ざり合って記憶され，混乱してしまいます。

今，次のルーチンがコンピュータの内部に入力されているとします。

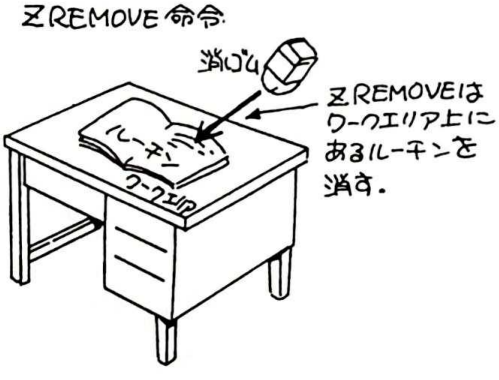
```
>ZP
F1      W !,"3+5=",3+5
F2      W !,"3-5=",3-5
F3      W !,"3*5=",3*5
F4      W !,"3/5=",3/5

>
```

ここでこのまま ZREMOVE 命令を使用せずに新しい行を入力してみましょう。

```
>G1      W !,"ﾊﾙﾙ ﾊ ｱｹﾎﾞﾉ."

>G2      W !,"ﾔｳﾔｳ ｼﾛｸ ﾅﾘﾕｸ."

>
```

ZPRINT してください。

```
>ZP
F1      W !,"3+5=",3+5
F2      W !,"3-5=",3-5
F3      W !,"3*5=",3*5
F4      W !,"3/5=",3/5
```



```
G1      W !,"ﾊﾙ ﾊ ｱｹﾎﾞﾉ."          ：今入力した行がここにあります
G2      W !,"ﾔｳﾔｳ ｼﾛｸ ﾅﾘﾕｸ."

>
```

後から入力した行が，すでにワークエリアに存在していたルーチンと一緒になっているのが分かったでしょうか？　コンピュータの本体内部（ワークエリア）には，同時に複数のルーチンは入りません。

新しいルーチンを入力するときは，その前に ZREMOVE 命令でワークエリア上のルーチンを消しておいてください。初心者はとかくこのことを忘れ，続けて別のルーチンを入力してしまうようです。

せっかく入力したプログラムを消してしまうのはもったいないと思われる方は「ルーチンを登録する」の節をよく読んでください。プログラムの蓄え方を解説します。

- ZREMOVE 命令はルーチンの内容を削除するのに用いる
- ZR　　はルーチン全体を削除する
- ZR 行指定　　は指定された行のみを削除する
- ZR 行指定1：行指定2　　は指定した範囲の行を削除する
- 新しいルーチンを作成する前には必ずZRでワークエリアのルーチンを削除しておくこと

5,ルーチンの修正——行を入れ替える

ルーチンのタイプミスなどのため，修正しなくてはならないことがしばしばあります。まず間違った行だけを ZREMOVE 命令で削除し，その直後に正しい行を入力します。

```
>ZP↵
A1      W 3+5          ：ルーチンを確かめました
A2      W 3-5          ：この行を修正しましょう

>ZR A1↵                ：まず，行A1を削除します
```

```
>X1[TAB]  W "3-5 ノ コタエ ハ "↵        ：次に，行X1を入力します

>ZP↵                                    ：確かめましょう
X1      W "3-5 ノ コタエ ハ "           ：行が入れ替わっています
A2      W 3-5

>
```

ここで，行を削除してから新しい行を入力する前に他の操作（とくに ZPRINT）をすると，入力した行は別の場所に挿入される可能性があります。挿入する場所の指定方法は「行を挿入する」のところで解説します。

- ルーチンの行を入れ替えるには ZREMOVE 命令で行を削除し，直後に正しい行を入力する

6,ルーチンの修正——行を挿入する

すでに存在する行を修正するのではなく，新しく行を追加する必要が生じることがあります。そのためには ZPRINT 命令を用います。

ZPRINT 命令で行が表示された直後に新しい行を入力します。するとその行は **ZPRINT 命令で表示された行のすぐうしろ** に挿入されます。そのあと続けて行を入力するとそれらも続いて後へ後へと挿入されます。

ZPRINT 命令で最後まで表示してから行を入力すると，新しい行はルーチンの最後に追加されていきます。

```
>ZP↵
A1      W 3+5
A2      W 3-5

>ZP A1↵                                 ：挿入する場所を指定します
A1      W 3+5                           ：行が表示されました

>B1[TAB]  W !,"ムカシ ムカシ アルトコロ ニ"↵      ：挿入したい行を入力します
```

```
>B2TAB   W !,"ｵｼﾞｲｻﾝ ﾄ ｵﾊﾞｱｻﾝ ｶﾞ"↵          ：この行も挿入します

>ZP↵                                          ：確認しましょう
A1      W 3+5
B1      W !,"ﾑｶｼ ﾑｶｼ ｱﾙﾄｺﾛ ｺﾞ"               ：挿入されています！
B2      W !,"ｵｼﾞｲｻﾝ ﾄ ｵﾊﾞｱｻﾝ ｶﾞ"             ：続けて挿入されています
A2      W 3-5


>
```

ルーチンの最後に行を追加したときは

```
>ZP↵
```

としてください。この直後に行を入力すると，たとえば次のようになります。

```
>ZP↵
A1      W 3+5
B1      W !,"ﾑｶｼ ﾑｶｼ ｱﾙﾄｺﾛ ｺﾞ"
B2      W !,"ｵｼﾞｲｻﾝ ﾄ ｵﾊﾞｱｻﾝ ｶﾞ"
A2      W 3-5


>C1TAB   W !,"ｵｼﾞｲｻﾝ ﾊ ﾔﾏ ﾍ ｼﾊﾞｶﾘ ｺﾞ"↵

>ZP↵
A1      W 3+5
B1      W !,"ﾑｶｼ ﾑｶｼ ｱﾙﾄｺﾛ ｺﾞ"
B2      W !,"ｵｼﾞｲｻﾝ ﾄ ｵﾊﾞｱｻﾝ ｶﾞ"
A2      W 3-5
C1      W !,"ｵｼﾞｲｻﾝ ﾊ ﾔﾏ ﾍ ｼﾊﾞｶﾘ ｺﾞ"          ：追加されています

>
```

ルーチンの先頭に行を挿入するのはやや面倒です。

先頭の行を ZREMOVE 命令でまず削除してから挿入すべき行と削除した行（前に先頭にあった行）を入力しなおしてください。

しかし実際には先頭に行を追加しなくてはならない事態はほとんどありません。というのは実際のルーチンでは，第１行目にいつも「ルーチンヘッダー」というそのルーチンのタイトルを入れておくからです。タイトルの前に行を挿入することはまずありません。

- 行を挿入するときはZPRINT命令で挿入すべき場所を表示する
- 新しく入力された行はZPRINT命令で表示された行の直後に挿入される
- ルーチンの最後に行を追加するときは，最後の行をZPRINTしてから追加する行を入力する
- ルーチンの先頭に行を追加するときに，第1行をZREMOVE命令で削除してから追加したい行を入力し，削除した行を再度入力する。

7,もっと便利な修正方法——エディタの使用

ZPRINT, ZREMOVE 命令を駆使することでルーチンを自由に編集することができますが，決して便利な方法とはいえません。たった1文字を修正するために1行をすべて入力しなおすことはかなり苦痛です。

これを解決するために「エディタ」というプログラムが用意されています。MUMPS によってエディタの使用方法が異なりますが，現在，SP-MUMPS 用に用意されている「^%エディタ」の使い方を解説しておきましょう。

いま次のルーチンの'オバアサン'を'オジイサン'に修正しましょう。

ここを変更します

```
>ZP↵
MOMO    W !,"ｱﾙ ﾋ ｵﾊﾞｱｻﾝ ｶﾞ"
TAROU   W !,"ｶﾜ ﾃﾞ ｾﾝﾀｸ ｼﾃｲﾙﾄ、"
        W !,"ｶﾜｶﾐ ｶﾗ ｵｵｷﾅ ﾓﾓ ｶﾞ、"
```

まず，1文字あけること

```
>X ^%↵
```

：'X' という命令を初心者が理解するのは少々難しいのでここではこのまま覚えてください[9]

「^」記号に注意。これはアッパー・キャレットとかサーカムフレクスとよばれる記号です。MUMPS ではこれからよくお目にかかります

としてください。

すると

編集行:

という表示が現れます。

エディタが呼び出されました，ここで修正したい行を指定してください。

形式は ZPRINT 命令などの指定方法と同じです。

編集行： MOMO↵

すると，この行が表示され

```
MOMO    W !,"ｱﾙ ﾋ ｵﾊﾞｱｻﾝ ｶﾞ"
```

さらに

変更したい文字：

というメッセージが表示され，カーソルがそこで止まっています。ここで，行の内容を修正できます。'オバアサン' を 'オジイサン' に修正しましょう。

変更したい文字： ﾊﾞｱ↵　　：修正すべき文字を入力します

とします。すると今度はその右に

変更後の文字：

というメッセージが現れるので，正しい文字（ここでは 'ジイ'）をタイプします。

変更後の文字： ｼﾞｲ↵　　：正しい文字を入力します

するとまたさらに

変更したい文字：

とたずねてくるので，ここでは↵キーだけを打ってください。もうこの行は修正しないという意味になります。エディタは

```
MOMO    W !,"ｱﾙ ﾋ ｵｼﾞｲｻﾝ ｶﾞ"     ：変更したという確認です
編集行：MOMO //
```

と表示します。これで行ＭＯＭＯの修正が終了しました。

この状態で他の行を指定すればその行を修正することができますし，もし何

■notes

9) X 命令は，XECUTE 命令といって「データを実行する」命令です。MUMPS には，他のコンピュータ言語にはない「間接指定」という「データを実行する」構文があります。これにはいくつかの型があり，XECUTE命令もそのひとつです。

この「間接指定」を上手に用いると非常に柔軟で強力なプログラムを書くことができます。MUMPS を特徴づける重要な構文です。

しかし，初心者向けではないので，この本では詳しくは解説しません。MUMPSの他の参考書を参照してください。

も指定せずに↵キーのみを打てば再度同じ行 MOMO を修正できます。

『編集行：』の質問に，ピリオド『．』を入力するとこのエディタは終了します。

```
編集行: MOMO // .↵        :「.」を入力します
>                          :エディタが終了しました
```

ZPRINT 命令で確認しましょう。

```
                    変更されています！
                          ↓
>ZP↵
MOMO    W !,"ｱﾙ ﾋ ｵｼﾞｲｻﾝ ｶﾞ"
TAROU   W !,"ｶﾜ ﾃﾞ ｾﾝﾀｸ ｼﾃｲﾙﾄ、"
        W !,"ｶﾜｶﾐ ｶﾗ ｵｵｷﾅ ﾓﾓ ｶﾞ、"

>
```

このエディタには，文字列を捜したり，まとめて置換したり，行を動かしたり，様々な機能がありますが，詳細は「SP-MUMPS プログラマーズガイド 第2章」を参照してください。なお

編集行:

の質問に，疑問符『？』を入力するとこのエディタの取り扱いの説明が表示されます。適宜利用してください。

◙エディタの使用方法

エディタの使い方をここで，まとめておきます。

①エディタの起動：…X ^%

②エディタの終了：…編集行：　でピリオド『．』を入力。

③文字の置換：……編集行：　で修正したい行を指定する。

　　変更したい文字：　で修正すべき文字列を入れる。

　　変更後の文字：　で正しい文字列を入力する。

④使用法の照会：……疑問符『？』を入力する。

8,ルーチンを登録する　ZSAVE/ZLOAD命令

コンピュータ内部のワークエリアには一時にひとつのルーチンしか記憶させておくことができません。

このため今までは，新しいルーチンを入力するごとに ZREMOVE 命令で古いルーチンをすべて消去してきました。しかし，後から同じ仕事をしようとしたときに，また最初からルーチンを入力しなおすのは実用的ではありません。

MUMPS にはルーチンをディスクに登録する機能があります。ディスクは録音テープと同じように磁気によって情報を記憶するので電源を切っても，そこに書き込まれた内容は消失することがありません。これに登録しておけば後日そのルーチンを呼び出して使用することができます。

登録はとても簡単で，ZSAVE という命令を用います。

ZS ルーチン名

とすると，そのときワークエリアにあるルーチンをルーチン名で指定した名前を付けてディスクに登録してくれます。

試してみましょう。

```
>ZP↵                          ：ワークエリアにあるルーチンを確認します
F1      W !,"3+5=",3+5
        W !,"3-5=",3-5
        W !,"3*5=",3*5
        W !,"3/5=",3/5

>
```

これに F1 という名前を付けて登録します。

```
>ZS F1↵        ：ZSAVE命令です
```

すると，ディスク装置がしばらくカチャカチャと音を立てて動きます。そして数秒後

```
>
```

という表示が画面に現れます。これで登録は終わりました。

登録されているかどうかはどうやって確認したらよいでしょうか。ワークエ

リア上のルーチンを消して，ディスクに登録したルーチンを呼び出すことができれば確かに登録されていることが分かります。

```
>ZR↵      ：ワークエリア上のルーチンを消去しました

>ZP↵      ：ZPRINTしても何も表示されません


>
```

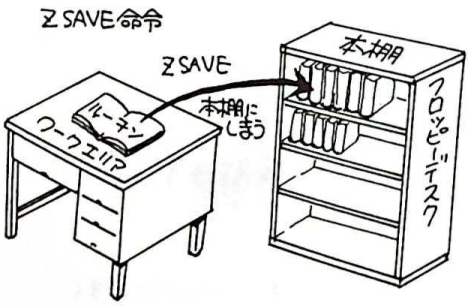
ここで，ZLOAD 命令というもうひとつの命令を覚えてください。これは，ディスクに登録されているルーチンをワークエリア上に持ってくる命令です。

```
>ZL F1↵       ：さきほどZSAVEで登録したルーチンF1を取り出
                します
```

多分ディスクがまたカチカチといって作動するはずです（ときにディスクバッファと呼ばれる機能が働いて実際にはディスクが作動せずに，同じ結果になることがあります。

ZPRINT で確かめます。

```
>ZP↵
F1      W !,"3+5=",3+5      ：確かにワークエリア上にルーチンが戻
        W !,"3-5=",3-5        ってきています
        W !,"3*5=",3*5
        W !,"3/5=",3/5


>
```



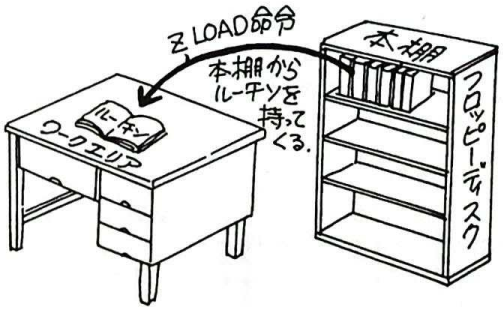
注意

ZSAVE 命令と ZLOAD 命令は，どちらもルーチンのコピーを作っているだけで，実際にルーチンがワークエリアとディスクの間を移動するわけではありません。したがってZSAVEしてもワークエリアにはルーチンが残っていますし，ZLOAD してもディスク上のルーチンは消えません。

ZSAVE 命令で指定した名前で別のルーチンがすでにディスクに登録してあった場合は，先に登録されていたルーチンは消えてしまいます。名前を間違えないようにしてください。

ZLOAD 命令でディスク上のルーチンを呼び出すと，そのときワークエリア上にあったルーチンは消えてしまいます。これはワークエリアには，一時にひとつのルーチンしか記憶させておくことができないからです。必要なルーチンは必ずディスクに登録してから別のルーチンを ZLOAD してください。

- ワークエリア上のルーチンをディスクに登録するには，ZSAVE 命令を用いる
- ディスクに登録されているルーチンを取り出すには，ZLOAD 命令を用いる
- ZSAVE 命令もZLOAD 命令もルーチンのコピーを作るだけで，これらの命令を実行しても，もとのルーチンはそのまま消えずに残っている。



9,ルーチン名の付けかた

ディスクに登録するルーチンの名前は ZSAVE 命令で指定したときに決まります。

```
>ZS ABC↵
```

とすれば，そのときにワークエリアにあるルーチンの名前は ABC になります。

```
>ZS XYZ↵
```

とすれば，名前は XYZ です。

ルーチンの名前としては行ラベルと同様に

①アルファベットの大文字

②数字

を用いることができます[10]。

しかし行ラベルと違い

③数字だけの名前は付けられません

また行ラベルと同様に

④最大長は 8 文字 notes 8) 参照

です。次のようなルーチン名は正しい名前です。

```
A   A10   R2D2   P3C0   ABCDEFG   Z2345678   HXI
```

次のようなルーチン名は正しくありません。

```
A%   10A   #@#   &H5   $CHAR   3333   AREA.COD   I_LOVE_YOU
```

ルーチン名は，ZSAVE 命令で決定され，行ラベルとは無関係です。しかし，ルーチンを表示したときに分かりやすくするため，ルーチンの第 1 行目の行ラベルとルーチン名は同一の名前にしておいてください。

notes

10) ルーチン名の第 1 文字には行ラベルと同様に「%」記号を付けることもできます。しかし，慣例上，「%」記号の付いたルーチン名は，「ユーティリティ」と呼ばれる特殊なルーチンにのみに用いられることになっています。普通は使用しないでください。

また，SP-MUMPS ではアルファベットの大文字の代わりに，小文字やカナ，漢字なども用いることができます。しかし，これらを用いると「標準 MUMPS」の範囲を超えてしまい，他の MUMPS システムと互換性がなくなるので使用しない方がよいでしょう。

また，これからルーチンを多数作ることになります。ルーチン名はできるだけ管理しやすい名前にしてください。

複数の人で同じコンピュータを使っていたり，さまざまな仕事に使用している場合には命名法を約束として決めておくことが必要です（Chapter3-12 参照）

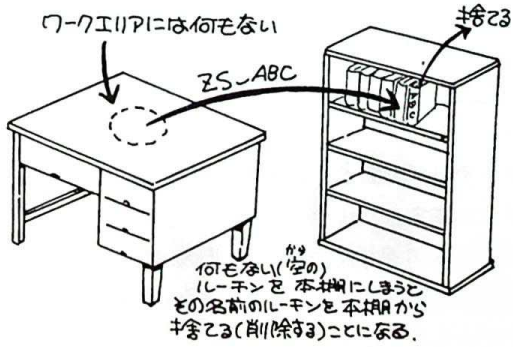
- ルーチン名は第1文字がアルファベットの大文字，第2文字以降はアルファベットの大文字または数字
- ルーチン名は最大8文字まで
- ルーチンの第1行目の行ラベル名とルーチン名は同一にしておくこと

10，ディスク登録ルーチンの削除

ディスクに登録してあるルーチンを削除する方法も知っておかねばなりません。削除するには，「空（から）のルーチン」を登録します。すなわち

```
>ZR↵
>ZS F1↵
```

ルーチンの削除



とすると，ワークエリアに何もない状態，つまり「空のルーチン」を F1 という名前で登録することになり，これは F1 を削除することになります。

登録されていない（削除された）ルーチンを ZLOAD しようとすると

<NOPGM> プログラムが定義されていません

というエラーになります。

```
>ZL F1↵

<NOPGM> プログラムが定義されていません
 0) *XECUTE*:: ZL F1                    ：エラーメッセージ
<NOPGM>*XECUTE*:::4:3 ZL F1
>
```

- ディスクに登録されているルーチンを削除するには，まずZRでワークエリアのルーチンを消去し，その「空のルーチン」を削除したいルーチンの名前でZSAVEする
- 登録されていないルーチンをZLOADしようとすると<NOPGM>というエラーを生じる

11，登録ルーチンの名前を調べる　DO命令

すでにどのようなルーチンが登録されているのかを調べたいことがしばしばあります。これには

```
>D ^%RD↵      ：DO命令です
```

「^」記号と「%」記号に注意してください

としてください。

するとモニターテレビ画面上に，ディスクに登録されているルーチンの名前がリストアップされます。

これは '%RD' という名前のユーティリティプログラムを実行しているのです。%RD は「ルーチンディレクトリ」(Routine Directory) の略で「ルーチン登録台帳」ということです。

ユーティリティプログラムというのは，MUMPS でプログラムを作成する際に，道具として使うために用意されたプログラムのことです。その名前には，一般に『%』記号が付けられています。

注意

D ^%RDという命令は%RDというプログラムを呼び出すので，そのときにワークエリア上にあったルーチンが消去されてしまうので注意してください

もしワークエリア上に必要なルーチンがあるときには，あらかじめそれをディスクに登録しておいてください

- ルーチン名はD ^%RDで調べることができる
- D ^%RDとすると，ワークエリア上のルーチンが消去されてしまう

12,ルーチンヘッダーの慣例

ZSAVE 命令でルーチンをディスクに登録するとき，ルーチンの名前は自由に付けることができます。しかし，画面に表示されているルーチンが何という名前でディスクに登録されているのかがすぐに分からないと困ることがあります。さらに，ルーチンを多数作成して行くと，どのルーチンが何のルーチンであったかを忘れてしまいます。

そこで，これは何というルーチンで，いつ誰が何のために作成したのかがわかるように，すべてのルーチンの第1行目の形式を整えておくと便利です。

この形式は「慣例」であって，MUMPS の文法規則ではありません。しかし，ほとんどのプログラマーがこの慣例にしたがってルーチンを書いているので，われわれもこれに従った方が何かと有利です。

慣例というのは次の通りです。

①ルーチン名の最初の2文字を統一すること

ルーチン名の最初の2文字を統一しておくと，そのルーチンは誰が作ったか，あるいは何の仕事のルーチンかをおおよそ見当付けることができ便利です。次に著者の使用しているルーチン群の名前を例としてあげておきます。

HX, HXI, HXIBC, HXO, HXOA, HXCWC	：検診データ処理のルーチン群
DI, DIA, DIC, DIX, DINIT, DINIT1	：File Manager というプログラムに含まれるルーチン群
YS01,YS02,YSLIST,YSREG,YSVAL	：イニシャル YS という人の作ったルーチン群

②ルーチンの第1行目の行ラベル名は，ルーチン名と同じ名前を用いること

ルーチンを ZPRINT 命令で表示したときにルーチン名が分かるので便利です。また，「ルーチンを登録する便利な方法」で解説しているＸＵＳ いう命令でルーチンを登録できるようになり，間違って登録する危険も減ります。

③ルーチンの第1行目は「注釈文」とし，次の形で，いつ誰が作成し，何をするものかを簡単に記載しておくこと

ルーチン名　[TAB]；作者，所属；作成年月日；簡単な説明

④第2行目も（必要なら第3行目以降も）「注釈文」とし，より詳しい説明を記載すること

実際例として，肥満度の計算ルーチンをあげます。

list 3-3

```
HIMAN3   ;YS,SAMPLE;1985.1.25;ﾋﾏﾝﾄﾞ ﾉ ｹｲｻﾝ
         ;
NYURYOKU R !,"ｼﾝﾁｮｳ = ",SHINCHO
         R !,"ﾀｲｼﾞｭｳ = ",TAIJU
KEISAN   S HYOJUN=SHINCHO-100*0.9
         S HIMANDO=TAIJU/HYOJUN-1*100
HYOJI    W !,"ﾋﾏﾝﾄﾞ = ",HIMANDO," %"
         Q
```

さて，第1行目と第2行目のセミコロン『；』は何でしょう。これは，この記号から右側の文字はプログラム作成者がメモとして書いておく「注釈文」であることを示しています。

MUMPS はルーチンの中にこの記号が現れると，それより右側は実行せず無視してしまいます。したがってセミコロンの右にはどんなことを書いてもよいのですが，先頭の第1，2行に限り上記の形式にそろえておきます。

第2行目のセミコロンは別に注釈文がある訳でもないので，必要ないようにも思われますが，ここでは第1行目を他の行から視覚的に分離して，見やすくするために挿入してあります。

セミコロン『；』は，コロン『：』と間違えやすいので注意してください。

こういう形式でルーチンの名前，所属，機能などを表わした部分を「ルーチンヘッダー」と呼んでいます。

ルーチンヘッダーのないルーチンを修正して，ヘッダーを作成するには，ルーチンの第1行目を削除して，ヘッダーといま削除した行を入力しなおします。これはルーチンの先頭に行を挿入する方法と同じです。

- セミコロン「；」を入れるとMUMPSはその右側を「注釈文」として，そこに書いてある内容を無視する
- ルーチンをディスクに登録するときには第1行目を次の形に整えておく
  ルーチン名TAB；作者，所属；作成年月日；簡単な説明
- ルーチンの第2行目も「；」を入れておく。
- ルーチンの先頭に行を挿入するには，第1行目を削除し，挿入したい行を入力した上で，削除した行を再度入力する。

◎ルーチンを登録する便利な方法

ZSAVE 命令でルーチンをディスクに登録するときに，思い違いやタイプミスなどで，誤ったルーチン名を指定してしまうという事故がときどき起こります。とくに，タイプミスしたルーチン名が，偶然すでに登録されているルーチンの名前であった場合には，前に登録されていたルーチンが消えてしまいます。そこで，いちいちルーチン名を指定せずにルーチンを登録する方法を解説します。ZSAVE 命令の代わりに

```
>X ^%US↵
```

とタイプしてください。するとワークエリア上のルーチンは，第1行目のラベル名でディスクに登録されます。そして，登録されると

[####] はディスクに登録されました。

というメッセージが表示されます（####はワークエリア上のルーチン名です）。

第1行目のラベル名に従って登録してくれるので，このラベル名さえ間違っていなければ，必要なルーチンを消してしまうという事故が防げます。ただしルーチンの第1行目の形式が前述の「ルーチンヘッダー」の慣例に従っていなくてはなりません。

もし〈UNDEF〉というエラーが生じた場合には，「file 4」を参照してください。この 'X ^%US' という命令の準備方法について解説してあります。

```
>ZP↵
SMPL01 ;YS,SAMPLE;1986.2.12;X ^%US ノレイ
       ;
F1     W !,"3+5=",3+5
       W !,"3-5=",3-5
       W !,"3*5=",3*5
       W !,"3/5=",3/5

>X ^%US↵                    ：登録します。

[SMPL01] はディスクに登録されました。
>
```

13, Z命令について

ZLOAD, ZSAVE, ZPRINT, ZREMOVE の4つの命令には 'Z' がついています。一方 WRITE, DO 命令には 'Z' はついていません。

これは ZLOAD, ZSAVE, ZPRINT, ZREMOVE の4つの命令が標準 MUMPS にはない命令だからです。このためこの本で解説している SP-MUMPS 以外の MUMPS ではこれら 'Z' のつく命令は，名前が違ったり使い方が異なっている可能性があります。たとえば，UCD-micro MUMPS[11] では，ディスクに登録してあるルーチンを削除するときには，空のルーチンを ZSAVE するのではなくて，ZDELETE という命令を用いて次のようにします。

```
>ZD ABC↵      ：ルーチン ABC を削除する
```

標準 MUMPS には21個の命令があります。しかしそれらの命令だけではルーチンを作ることさえできないので，ほとんどすべてのメーカーは標準外の命

■notes

11) UCD-micro MUMPSはカリフォルニア大学デービス校で作られたソフトウェアです

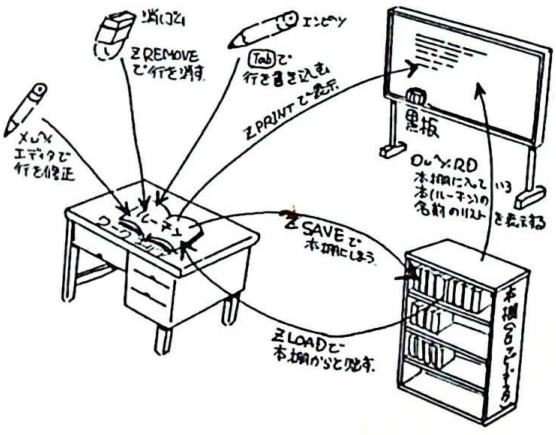
令を用意しています。標準 MUMPS のきまりでは，名前が混乱するのを避けるため，標準外の命令には ' Z ' をつけることが義務づけられており，ZLOAD, ZSAVE といった名前になっています。そして標準の命令は頭文字1字に省略され，標準外の ' Z ' 命令はZを含めて2文字に省略されます。

ルーチンを他の MUMPS に移すとき，ルーチン中に ' Z ' 命令があれば，そこは「標準語」ではないという意味になります。もし ' Z ' 命令がなければ「標準語」のみで書いてあるルーチンということになり，容易に他の MUMPS に移植することができます。

また，ZLOAD, ZSAVE, ZPRINT, ZREMOVE などはルーチンを作ったり保存したりするのに必要な命令であって普通のルーチンの中にはまず出現しないのでこれらが標準命令に含まれていなくてもまったく問題ありません。

ユーザーから見ると，MUMPS のメーカーが異なるごとにルーチンの作り方が大きく違うと困ります。しかし幸いルーチンを編集するための命令については，上記のような細かい違いはあるものの，多くのメーカーが SP-MUMPS と同じ働きで，同じ名前の ' Z ' 命令を用意しておりユーザーが大変に困るといった事態はありません。

- 頭文字にZのつく命令は標準MUMPS以外の命令である
- Zのつく命令はZを含めて2文字に省略できる
- ルーチン中にZのつく命令があると，そのルーチンそのままでは他のMUMPSシステムに移すことができない



Step 2

プログラミングMUMPS

ここでは職員名簿という例題を用いて，データの基本操作のプログラム（ルーチン）について説明します。

Chapter4ではMUMPSにデータを覚えさせ，演算させる方法を説明します。

Chapter5では具体的な内容としてデータを登録する方法を示しましょう。

Chapter6ではリストする条件に合うデータを捜すことにします。

Chapter 4

データを記憶する

電卓には，計算結果を記憶させておく機能を持ったものがあります。電卓にはたかだか数個のデータしか記憶させられませんが，MUMPS にはとても大量のデータを記憶させておくことができます。

MUMPS にはデータを記憶し，それを用いて様々な処理に用いるために変数というものが用意されています。

この章では変数のもっとも基本的な概念と，キーボードからデータを入力する命令について学びます。

1,データのメモリと変数

SET命令

MUMPS にデータを記憶させるには，SET という命令を用います。次の通りにタイプしてください。

```
>S X=1234↵
```

としてください。これで，1234というデータをXという名前をつけて記憶させました。言い換えると，変数Xに値1234をセットしたということです。

変数Xに入っている値を調べてみましょう。

WRITE 命令を用います。

```
>W X↵
1234                    ：変数Xの値を表示します
                        ：変数Xの値です
>W X," ",X↵             ：もう一度表示します
1234 1234
>W X*15↵
18510                   ：Xの15倍の値です
>
```

このように，いったん記憶させたデータはXという名前で何回でも取り出し，計算に使用することができます。

このXのことを「変数」といい，変数にデータを記憶させる命令を SET 命

```
>S YY=555↵          ：変数YYに555という値をセットします

>W YY↵
555                 ：YYの値です
>S ZZ=YY+123↵       ：YYを計算に出い，ZZという新しい変数をセットしました

>W ZZ↵
678
>S ZZ=YY*YY*YY↵

>W ZZ↵
170953875
>
```

令といいます。

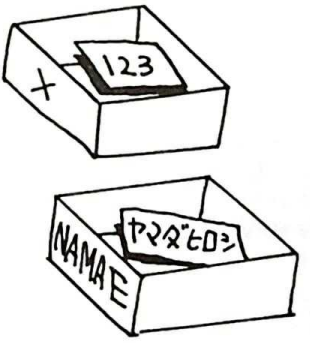
いろいろな場合を試して SET 命令と変数に慣れてください。

上記の３つめの例ではすでに678というデータが入っていました。

そのZZに，あらためてYY＊YY＊YYすなわち，170953875というデータを入れなおしています。その結果古いデータは捨てられて新しいデータが記憶されていることに注意してください。

変数を「箱」にたとえるなら，そこにはデータを書いた「カード」が入ります。「カード」は常に１枚しか入らず，新しい「カード」を入れると古い「カード」が自動的に捨てられます。

もう少し続けましょう。



←変数はデータを書きこんだ「カード」を入れる箱です。ひとつの「箱」にはいつも１枚のカードが入っています。

←SET命令で指定した名前の「箱」がなくても自動的に「箱」が作られます。

```
>S NAMAE="ﾊﾔﾐ ﾕｳ"↵

>W NAMAE↵
ﾊﾔﾐ ﾕｳ
>S BOOK="Give Me Shelter!"↵

>W BOOK↵
Give Me Shelter!
>
```

この2例では，数値ではなく引用符『 』で囲まれた文字の並び（文字列）の内容が，そのまま NAMAE と BOOK という変数に記憶されていることに注意してください。SET 命令は一般に

S　変数＝数値
S　変数＝引用符「 」ではさまれた文字列
S　変数＝式

の形をしています。変数の名前は自由につけられますが，使用できる文字は

①アルファベットの大文字
②数字

です。ただし

③先頭はアルファベットの大文字

でなくてはなりません。[1]いずれにしても

④最大長8文字まで

です。[2]

たとえば，次のような変数名は正しいものです。

```
A   A10   R2D2   C3PO   ABCDEFG   A234567
```

■notes

1) 第1字に限って「%」記号も使用できます。しかしこれは慣例として、ユーティリティプログラムという特殊なルーチンに使用することになっており，普通には使いません。
また，SP-MUMPSではアルファベットの大文字の代わりに，アルファベットの小文字，カナや漢字も用いることができます。しかし，標準MUMPSの範囲を越えてしまうので，他のMUMPSにデータやプログラムを移植できなくなる可能性があります。したがって，本文のとおりの制限に従っていた方がよいでしょう。

2) 標準MUMPSの規定では，行ラベル（Chapter3－1参照）やルーチン名（Chapter3－9参照）変数名には最大15文字まで使用できます。しかし，MUMPSが認識するのは8文字までとなっています。このため，最大8文字と覚えておいた方がよいでしょう。

次のようなものは正しくありません。

```
A%    10A    #@#    &H5    $CHAR    3333
AREA.COD    I_LOVE_YOU
```

この規則はルーチン名の規則とまったく同じです。

- SET命令を用いてデータを変数に記憶させることができる
- SET命令の形式：S　変数＝数値　S　変数＝"文字列"
  S　変数＝式
- 変数の名前には第1文字が「%」記号，またはアルファベットの大文字。第2文字以降はアルファベットの大文字または数字である
- 変数の名前の最大長は8文字

2,SET命令を用いたルーチンを作る

次に SET 命令を用いたルーチンの例を挙げてみましょう。

list 4-1

```
HIMAN   ;YS,SAMPLE;1985.1.25;ﾋﾏﾝﾄﾞ ﾉ ｹｲｻﾝ
        ;
SET     S SHINCHO=160
        S TAIJU=81
KEISAN  S HYOJUN=SHINCHO-100*0.9
        S HIMANDO=TAIJU/HYOJUN-1*100
HYOJI   W !,"ﾋﾏﾝﾄﾞﾞ = ",HIMANDO," %"
        Q
```

注意！　これは濁点「ﾞ」です
引用符「"」とは異なります

このルーチンについて少し説明しましょう。

1行目と2行目はルーチンヘッダーです。1行目の行ラベルはルーチンの名前，そのあとに作者，所属，作成年月日，説明が注釈として入れてあります。

実際の命令は3行目から始まります。3行目と4行目は次のとおりです。

```
SET       S SHINCHO=160
          S TAIJU=81
```

これは SHINCHO という変数に160を，TAIJU という変数に81という値をセットせよという命令です。5行目は

```
KEISAN    S HYOJUN=SHINCHO-100*0.9
```

で，SHINCHO という変数の値を用いて SHINCHO－100*0.9 という式を計算し，その結果の54を HYOJUN という別の変数にセットしています。

計算順序を覚えていますか。MUMPS では加減乗除に優先順位がないので，SHINCHO－100*0.9 は (160－100)*0.9 と解釈されます。

6行目では，さらにその HYOJUN と TAIJU から HIMANDO というまた別の変数を作っています。

```
S HIMANDO=TAIJU/HYOJUN-1*100
```

そして WRITE 命令で HIMANDO の値を表示しています。

最後の行に『Q』という1文字の命令がありますが，これは QUIT 命令といって，このルーチンはここで終了という意味になります。

最後の行の QUIT 命令に限り省略できるので，今までは使用しませんでした。これからは終了を明確にするために省略せずに使います。

このルーチンはお分かりの通り，身長と体重から標準体重，肥満度を計算しているルーチンです。入力して実行してください。

```
>D HIMAN↵

ﾋﾏﾝﾄﾞ = 50 %
>
```

◙プログラムの誤りを見つけるためのヒント

もし，エラーメッセージが出てしまったら，どこかに間違いがあります。じっくり調べてください。

①必要な空白は確かに入っているか。余分な空白はないか

②コンマ『,』は間違いないか。ピリオド『.』やアポストロフィ『'』と間違えていないか

③引用符も正しく入っているか

④変数名をタイプミスしていないか

⑤ TAB は正しく入れたか

⑥このルーチンを実行する DO 命令は正しかったか

見た目には正しいのにどうしてもエラーとなる場合にはもういちどタイプしなおしてください。

制御文字という目に見えない文字が入っていてそれがエラーのもとになっていることがあるからです。

- QUIT命令は，ルーチンの終わりを示す
- ルーチンの最後にあるQUIT命令は省略することができる

3,変数の内容を調べる

変数の内容を調べるには，WRITE 命令でその変数の名前を指定します。

```
>W TAIJU↵
81
>W SHINCHO↵
160                ：MUMPSは以前にセットしたデータを
>W HYOUJUN↵          すべて覚えています
54
>W X↵
1234               ：別のデータが表示されるかもしれませ
>W NAMAE↵            ん。それはあなたが以前別のデータを
ﾊﾔﾐ ﾕｳ               セットしたからです
>
```

もしかしたら

<UNDEF> 定義されていない変数が参照されました

というエラーメッセージが表示されるかもしれません。これは WRITE 命令で指定した変数が存在しなかったという意味です。

さあここで，単にWとだけタイプしてください，何が起こりますか。

```
>W↵                ：引数のないWRITE命令

BASHO="ｱﾗﾓｺﾞｰﾄﾞ"
HIMANDO="50"
HIZUKE="1945/7/16"
HYOJUN="54"
NAMAE="ﾊﾔﾐ ﾕｳ"
SHINCHO="160"      ：実際にモニターテレビ画面に表示され
TAIJU="81"           る内容は少し異なるかもしれません
X="1234"
YY="55"
ZZ="ABC"
>
```

そのときにセットされている変数がすべて表示されました。WRITE 命令は変数も式も，何も指定しないと（引数を何もタイプしないと），そのときに存在する変数のリストを表示してくれます。

- 定義されていない変数の値を表示したり，計算に使おうとすると，「<UNDEF>定義されていない変数が参照されました」というエラーとなる
- 引数なしのWRITE命令は，そのときにセットされている変数のリストを表示する

4,データを削除する

KILL命令

ルーチンを削除するには ZREMOVE 命令を用いました。MUMPS には変数を削除する命令も用意されています。その命令を KILL 命令といいます。

```
>K X↵
```

これで変数Xが削除されました。

```
>K YY,ZZ,TAIJU↵
```

YY, ZZ, TAIJU という３つの変数を削除しました。

WRITE 命令で調べてみましょう。

```
>W↵
BASHO="ｱﾗﾓｺﾞｰﾄﾞ"
HIZUKE="1945/7/16"      ：変数X, YY, ZZ, TAIJU が
HYOJUN="54"               表示されません
NAMAE="ﾊﾔﾐ ﾕｳ"
SHINCHO="160"
>
```

KILLで削除した変数をWRITEで表示しようとするとエラーになります。

```
>W TAIJU↵

<UNDEF> 定義されていない変数が参照されました
 0) *XECUTE*:: W TAIJU
<UNDEF>*XECUTE*:::4:0 W TAIJU
```

MUMPS は KILL 命令で削除しない限り，変数をいつまでも記憶しています。ところがコンピュータの記憶装置のワークエリアには限りがあり，あまり沢山のデータをためておくといつか一杯になり，あふれてしまいます。すると

```
<PGMOV> メモリが不足しています
```

というエラーが生じ，それ以上仕事ができなくなります。

このため，不必要な変数は KILL 命令で削除しておくことを勧めます。

なお ZREMOVE 命令で行ラベルを指定しないとルーチン全体を消してしまったように，変数を指定しない KILL 命令は，ワークエリア上の変数をすべて削除します。

```
>K↵        ：変数を指定しないKILL命令

>W↵        ：セットされている変数をすべて表示するWRITE命令
           ：何も表示されません

>
```

- KILL命令は指定された変数を削除する
- KILLされた変数を表示しようとすると，<UNDEF> というエラーになる
- 変数を指定しないKILL命令はワークエリア上の変数をすべて削除する
- MUMPSはKILL命令で消さない限り，変数をいつまでも記憶している
- セットする変数が多過ぎると <PGMOV> というエラーになり，仕事ができなくなる

5,いろいろな値を計算する　READ命令

前節のルーチンは身長 160 cm 体重 81 kg の人の肥満度を計算するように作ってあります。これ以外の値については計算することができません。何回実行してもいつも同じ結果しか表示されません。

```
>D HIMAN↵

ﾋﾏﾝﾄﾞ = 50 %
>D HIMAN↵

ﾋﾏﾝﾄﾞ = 50 %
>
```

リスト 4-1 のルーチンの第 3 行目と第 4 行目を変更すれば，異なった身長，体重の人の肥満度を計算することができます。

あなたの身長と体重からあなたの肥満度を計算するように修正してください。

しかしこのように，いちいちルーチンを変更するのは非実用的な方法です。

MUMPS にはルーチンを実行するごとに変数にいろいろな値をセットすることのできる READ という命令があります。ここで READ 命令についてふれます。

まず READ 命令を理解するために次のとおりにタイプしてください。

```
>R NAMAE↵
```

すると MUMPS は何も表示せず，ただモニターテレビの画面の次の行の左端にカーソルを点滅させているだけです。「＞」記号も出ません。

ここで，イワサキとタイプして最後に↵を打ってください。

```
ｲﾜｻｷ↵   ：今，イワサキとタイプしました
>
```

下線はあなたがタイプしたというしるしです

今度は「＞」記号が出てきました

何が起こったのでしょう？

WRITE 命令で変数 NAMAE の値を調べてください。

```
>W NAMAE↵
ｲﾜｻｷ          ：これはMUMPSが表示しました
>
```

イワサキという値が表示されました。READ 命令を実行した直後にキーボードからタイプしたデータが変数 NAMAE にセットされているのです。何回か同じことを試してみましょう。

```
>R HIZUKE↵
1945/7/16↵    ：下線はあなたのタイプした内容を示しています
>W HIZUKE↵
1945/7/16     ：下線のついていないのはMUMPSの表示した内
                容です
>R BASHO↵
ｱﾗﾓｺﾞｰﾄﾞ↵
>W BASHO↵
ｱﾗﾓｺﾞｰﾄﾞ
>
```

READ 命令を実行すると，MUMPS はユーザーがキーボードからデータを入力するのを待っています。

ユーザーがデータの最後に↵キーを打つと，それまでに入力されたデータがREAD 命令で指定された変数に自動的にセットされます。

これを用いると肥満度の計算もずいぶん便利になりそうです。

list 4-2

```
HIMAN2   ;YS,SAMPLE;1986.2.12;ﾋﾏﾝﾄﾞ ﾉ ｹｲｻﾝ
         ;
NYURYOKU W !,"ｼﾝﾁｮｳ = "
         R SHINCHO
         W !,"ﾀｲｼﾞｭｳ = "
         R TAIJU
KEISAN   S HYOJUN=SHINCHO-100*0.9
         S HIMANDO=TAIJU/HYOJUN-1*100
HYOJI    W !,"ﾋﾏﾝﾄﾞ = ",HIMANDO," %"
         Q
```

このルーチンを入力して，実行してみましょう（ルーチンを入力する前にZR 命令を忘れないこと！）。

```
>D HIMAN2↵
ｼﾝﾁｮｳ = 150↵                          ：下線はあなたがタイプした文字です
ﾀｲｼﾞｭｳ = 60↵
ﾋﾏﾝﾄﾞ = 33.33333333333333 %          ：これはMUMPSが表示しました
>D HIMAN2↵

ｼﾝﾁｮｳ = 162↵
ﾀｲｼﾞｭｳ = 45↵
ﾋﾏﾝﾄﾞ = -19.35483870967742 %
>
```

これであらゆる人の肥満度を計算することができるようになりました。

- READ命令を用いるとルーチンの実行ごとに，キーボードからデータを入力することができる
- R HENSUは入力されたデータを変数HENSUにセットする

6,READ命令の代表的パターンを覚えよう

ところでリスト 4-2 のルーチンの第 3 行目と第 4 行目に注目してください。

```
W !,"ｼﾝﾁｮｳ = "
R SHINCHO
```

これは画面に 'シンチョウ=' と表示してから次に変数 SHINCHO にデータを読み込んでいます。

原理的には 'シンチョウ=' という表示がなくても変数にデータを読み込むことは可能です。しかしこの表示がないと MUMPS が今，何のデータの入力を待っているのかユーザーには分かりません。

したがって実用的なルーチンでは，ほとんどの場合ユーザーに対して次は何のデータを入力してくださいというメッセージを表示してから入力を待ちます（このメッセージのことを「プロンプト」または「入力指示」と呼んでいます)。

プロンプトを表示するには前節の例のように WRITE 命令と READ 命令を組み合わせて使用してもよいのですが，MUMPS ではこれを READ 命令のみで行うことができます。次の例を見てください。

list 4-3

```
HIMAN3   ;YS;SAMPLE;1986.2.12;ﾋﾏﾝﾄﾞ ﾉ ｹｲｻﾝ
         ;
NYURYOKU R !,"ｼﾝﾁｮｳ = ",SHINCHO          ：ココに注目！
         R !,"ﾀｲｼﾞｭｳ = ",TAIJU           ：ココに注目！
KEISAN   S HYOJUN=SHINCHO-100*0.9
         S HIMANDO=TAIJU/HYOJUN-1*100
HYOJI    W !,"ﾋﾏﾝﾄﾞ = ",HIMANDO," %"
         Q
```

この QUIT 命令はルーチンの終わりを示す命令です

この第 3 行目と第 4 行目の形が最も頻繁に使用される READ 命令の形です。

```
R !,"ﾌﾟﾛﾝﾌﾟﾄ",HENSU        ：この形です
```

結局

R！" は W！ の代わりになり

R"メッセージ" は W"メッセージ" の代わりになる

のです。

- READ命令はW！とW“メッセージ”の機能を代行できる
  R！はW！の，R "ABC" はW "ABC" の、それぞれ代わりになる
- READ命令の典型的なパターンは
  R！，"プロンプト"，HENSU　である
- 入力指示のメッセージ（プロンプト）を表示することにより，ユーザーは何のデータを入力すべきか分かりやすい

■7,READ命令を用いた簡単なルーチン

1ドルを円に換算する

READ 命令を用いるといろいろな仕事ができます。その例としてドルを円に換算するルーチンをあげます。リスト4-4は単に１ドルが何円に当たるか，換算したいのは何ドルかを入力すると，何円になるかを表示するルーチンです。

list 4-4

```
KANSAN  ;YS,SAMPLE;1986.1.15;「ﾄﾞﾙ」ｶﾗ「ｴﾝ」ﾍﾉ ｶﾝｻﾝ
        ;
RATE    R !,"1 ﾄﾞﾙ ﾊ ﾅﾝ ｴﾝ ? ",A
DOLLAR  R !,"ﾅﾝﾄﾞﾙ ｦ ｶﾝｻﾝ ｼﾏｽｶ ? ",B
KEISAN  S EN=A*B
HYOUJI  W !,B," ﾄﾞﾙ ﾊ ",EN," ｴﾝ ﾃﾞｽ。"
        Q
```

実行します。

```
>D KANSAN↵

1 ﾄﾞﾙ ﾊ ﾅﾝ ｴﾝ ? 186↵
ﾅﾝﾄﾞﾙ ｦ ｶﾝｻﾝ ｼﾏｽｶ ? 159↵
159 ﾄﾞﾙ ﾊ 29574 ｴﾝ ﾃﾞｽ。
>
```

2 金種計算

もうひとつ READ 命令を用いた簡単なルーチンの例として「金種計算」のルーチンをあげておきます。

「金種計算」というのは，ある金額のお金を支払うときに，どの紙幣や硬貨を何枚ずつ払えば，紙幣と硬貨の枚数を最小限にできるかを計算するものです。たとえば，53762円を払うには

一万円札　5枚	五百円硬貨　1枚	十円硬貨　1枚
五千円札　0枚	百円硬貨　2枚	五円硬貨　0枚
千円札　3枚	五十円硬貨　1枚	一円硬貨　2枚

を用いるのが枚数としては最少限になります。

計算の方法は，「与えられた金額を紙幣または硬貨の金額で割り，小数を切り捨てた値をその紙幣または硬貨の枚数とする。そしてその残りを額の小さな紙幣または硬貨で払う」ということです。文章にすると何か難しそうですが，よく考えると当たり前のことです。

人間はこれを直感的に計算しているのです。

ここで小数の切り捨てと，割り算の余りの計算が必要です。MUMPS にはこれを行う便利な記号があります。

『¥』記号[3]は割り算で小数点以下を切り捨てた値を計算します（整数除算といいます）。

```
>W 9¥4↵
2               ：9を4で割ると，2余り1
>W 14¥4↵
3               ：14を4で割ると，3余り2
>W 16¥4↵
4               ：16を4で割ると，4余り0
>
```

『#』記号は割り算の余りを計算します（「法」とか「モジュロ」といいます）。

```
>W 9#4↵
1               ：9を4で割ると，2余り1
>W 14#4↵
2               ：14を4で割ると，3余り2
>W 16#4↵
0               ：16を4で割ると，4余り0
>
```

■notes

3) 『¥』記号は，コンピュータの端末装置によっては『＼』記号になっている場合がありますが，同じ文字として扱われます

さて計算ルーチンです。少々長いですが，構造はいたって簡単です。

list 4-5

```
KINSHU  ;YS,EXAMPLE;1986.1.15;ｷﾝｼｭ ｹｲｻﾝ
        ;
KINGAKU R !,"ｷﾝｶﾞｸ ? ",KINGAKU
10000   S N10000=KINGAKU¥10000
        S AMARI=KINGAKU#10000
5000    S N5000=AMARI¥5000
        S AMARI=AMARI#5000
1000    S N1000=AMARI¥1000
        S AMARI=AMARI#1000
500     S N500=AMARI¥500
        S AMARI=AMARI#500
100     S N100=AMARI¥100
        S AMARI=AMARI#100
50      S N50=AMARI¥50
        S AMARI=AMARI#50
10      S N10=AMARI¥10
        S AMARI=AMARI#10
5       S N5=AMARI¥5
1       S N1=AMARI#5
KEKKA   W !,"10000 ｴﾝ ｻﾂ  = ",N10000," ﾏｲ"
        W !," 5000 ｴﾝ ｻﾂ  = ",N5000," ﾏｲ"
        W !," 1000 ｴﾝ ｻﾂ  = ",N1000," ﾏｲ"
        W !,"  500 ｴﾝ ﾀﾞﾏ = ",N500," ﾏｲ"
        W !,"  100 ｴﾝ ﾀﾞﾏ = ",N100," ﾏｲ"
        W !,"   50 ｴﾝ ﾀﾞﾏ = ",N50," ﾏｲ"
        W !,"   10 ｴﾝ ﾀﾞﾏ = ",N10," ﾏｲ"
        W !,"    5 ｴﾝ ﾀﾞﾏ = ",N5," ﾏｲ"
        W !,"    1 ｴﾝ ﾀﾞﾏ = ",N1," ﾏｲ"
        Q
```

このルーチンは短くはありませんが，慎重に入力して試してみましょう。

```
>D KINSHU↵

ｷﾝｶﾞｸ ? 53762↵
10000 ｴﾝ ｻﾂ  = 5 ﾏｲ
 5000 ｴﾝ ｻﾂ  = 0 ﾏｲ
 1000 ｴﾝ ｻﾂ  = 3 ﾏｲ
  500 ｴﾝ ﾀﾞﾏ = 1 ﾏｲ
  100 ｴﾝ ﾀﾞﾏ = 2 ﾏｲ
   50 ｴﾝ ﾀﾞﾏ = 1 ﾏｲ
   10 ｴﾝ ﾀﾞﾏ = 1 ﾏｲ
    5 ｴﾝ ﾀﾞﾏ = 0 ﾏｲ
    1 ｴﾝ ﾀﾞﾏ = 2 ﾏｲ

>
```

回演算子

加減乗除の＋，－，＊，/や，整数除算の¥，モジュロの＃のことを「演算子」と呼びます。演算子は，ふたつの値の間にあって，そのふたつの値を用いて決まった計算をし，何かの値を作り出す働きがあります。

たとえば，３＋５という式では，演算子『＋』が，３と５という値を加え，８という値を作り出します。

- 「¥」記号は，割り算の結果から小数点以下を切り捨てた値を計算する
- 「＃」記号は，割り算の結果の余りを計算する

Chapter 5

職員名簿を作る

MUMPS が得意とし，もっともよく使われる仕事は，大量のデータを蓄えることです。これから「職員名簿」という例をもとにして，データを蓄え，それを様々な形で利用する方法を解説します。

1,キーボードでタイプしたデータを蓄える

ここで作る「職員名簿」に書き込むデータの項目は

①名前
②性別
③年齢
④所属部署

以上です。もっといろいろなデータ項目が欲しいかもしれませんが，今はこれだけで我慢しましょう。

さてキーボードでタイプしたデータを蓄えるというと何か難しそうですが，実は非常に簡単です。

list 5-1

```
MB001   ;YS,ｼｮｸｲﾝﾒｲﾎﾞ;1985.2.19;ﾃﾞｰﾀ ﾆｭｳﾘｮｸ
        ;
START   R !,"ﾅﾏｴ ..... : ",NAMAE
        R !,"ｾｲﾍﾞﾂ .... : ",SEI
        R !,"ﾈﾝﾚｲ ..... : ",TOSHI
        R !,"ｼｮｿﾞｸ･ﾌﾞｼｮ : ",BUSHO
        Q
```

これで終わりです。

さて，実行してください。名前，性別，年齢，所属をたずねてきます。データを入力しましょう。

```
>D MB001

ﾅﾏｴ ..... : ｲﾜｻｷ ﾋﾛﾐ
ｾｲﾍﾞﾂ .... : F                    : Fは女性(Female)のことです
ﾈﾝﾚｲ ..... : 26                     男性はM(Male)です
ｼｮｼﾞｸ･ﾌﾞｼｮ : ｺｳﾎｳｼｼﾝ
>
```

入力したデータはそれぞれ NAMAE, SEI, TOSHI, BUSHO という4つの変数に記憶されます。WRITE 命令で確かめてください。

```
>W NAMAE
ｲﾜｻｷ ﾋﾛﾐ
>W SEI
F
>W TOSHI
26
>W BUSHO
ｺｳﾎｳｼｼﾝ
>
```

次へ移る前に，ルーチンをディスクに登録しましょう。

```
>ZS MB001
```

または

```
>X ^%US
```

です。

•READ命令を用いるとデータを変数に蓄えるルーチンを作ることができる。

2,多人数のデータを蓄える

しかし，このルーチンでは見ての通りデータは1人分しか入りません。2人目のデータを入力すると前に入力したデータは消えてしまいます（試してください)。

そこで次に5人のデータを入力できるようにしましょう。

list 5-2

```
MB002   ;YS,ｼｮｸｲﾝﾒｲﾎﾞ;1985.2.19;ﾃﾞｰﾀ ﾆｭｳﾘｮｸ (5 ﾆﾝ ﾌﾞﾝ ﾉ ﾃﾞｰﾀ)
        ;
HITORI  R !,"ﾅﾏｴ ..... : ",NAMAE1
        R !,"ｾｲﾍﾞﾂ .... : ",SEI1
        R !,"ﾈﾝﾚｲ ..... : ",TOSHI1
        R !,"ｼｮｿﾞｸ･ﾌﾞｼｮ : ",BUSHO1
FUTARI  R !,"ﾅﾏｴ ..... : ",NAMAE2
        R !,"ｾｲﾍﾞﾂ .... : ",SEI2
        R !,"ﾈﾝﾚｲ ..... : ",TOSHI2
        R !,"ｼｮｿﾞｸ･ﾌﾞｼｮ : ",BUSHO2
SANNIN  R !,"ﾅﾏｴ ..... : ",NAMAE3
        R !,"ｾｲﾍﾞﾂ .... : ",SEI3
        R !,"ﾈﾝﾚｲ ..... : ",TOSHI3
        R !,"ｼｮｿﾞｸ･ﾌﾞｼｮ : ",BUSHO3
YONIN   R !,"ﾅﾏｴ ..... : ",NAMAE4
        R !,"ｾｲﾍﾞﾂ .... : ",SEI4
        R !,"ﾈﾝﾚｲ ..... : ",TOSHI4
        R !,"ｼｮｿﾞｸ･ﾌﾞｼｮ : ",BUSHO4
GONIN   R !,"ﾅﾏｴ ..... : ",NAMAE5
        R !,"ｾｲﾍﾞﾂ .... : ",SEI5
        R !,"ﾈﾝﾚｲ ..... : ",TOSHI5
        R !,"ｼｮｿﾞｸ･ﾌﾞｼｮ : ",BUSHO5
        Q
```

これで５人分のデータが入るようになりました。

１番目の人の名前は変数 NAMAE1 に，性別は SEI1 に，年齢は TOSHI1 に，所属は BUSHO1 に格納されています。

５番目の人の名前は変数 NAMAE5 に，性別は SEI5 に，年齢は TOSHI5 に，所属は BUSHO5 に格納されています。同様に，ルーチンの行数を増やしていけば，何人のデータでも蓄えることができます。試してください。

3,変数に添え字をつける

しかしこれでもまだ実用的なルーチンとはいえません。20人のときはどうしますか？　100人ならどうしますか？　2000人が相手では？

ここで新しい変数の形式を覚えてください。「添え字付き変数」です。

たとえば今まで使用してきた

NAMAE1，　NAMAE2，　NAMAE5，　NAMAE10，

SEI2，　SEI4，　TOSHI3，　TOSHI27

といった変数を

```
NAMAE(1),NAMAE(2),NAMAE(5),NAMAE(10),
SEI(2),SEI(4),TOSHI(3),TOSHI(27)
```

というように替えることができます。変数名の数字の部分をカッコで囲みました。このカッコ（ ）で囲まれた部分を「添え字」と呼んでいます。

添え字付き変数は，変数名が同じでも，添え字の値が異なると別の変数として扱われます。つまり添え字のない変数と別のものとして扱われます。試してみましょう。

```
>K↵          ：ワークエリア上の変数をすべて削除します

>S A(2)="ABC",A(7)="PQR",A2="XYZ",A="SSS"↵

>
```

4つの変数をセットします
コンマに注意！

引数なしの WRITE 命令でこれらの変数を表示しましょう。

```
>W↵

A="SSS"
A(2)="ABC"
A(7)="PQR"
A2="XYZ"
>
```

A, A(2), A(7),A2 という別々の4つの変数として表示されました。

変数の名前の数字部分をカッコに入れて添え字にしただけでは何の意味もありません。添え字を用いる利点は，添え字の部分に別の変数を入れることができるということです。すなわち

```
TOSHI(1), TOSHI(2), TOSHI(3),
TOSHI(5), TOSHI(10), TOSHI(1234)
```

といった変数をすべて

```
TOSHI(X)
```

で表すことができるようになります。

X＝1　のときTOSHI(X)はTOSHI(1)のこと
X＝2　のときTOSHI(X)はTOSHI(2)のこと
X＝5　のときTOSHI(X)はTOSHI(5)のこと
X＝1234のときTOSHI(X)はTOSHI(1234)のこと

になるのです。変数の中に別の変数が入ってしまいました。

さらに，添え字には計算式を入れることもできます。たとえば

X＝4　Y＝4　NO＝1

という３つの変数があったとき

```
TOSHI(X)　TOSHI(Y)　TOSHI(NO+3)
```

はすべて

```
TOSHI(4)
```

のことを表し，同じ変数としてみなされます。

添え字付き変数は，**添え字の値が何か**ということで区別されます。

添え字として用いられている変数が何かとか，定数であるか，式であるかということでは区別されません。

一方

```
TOSHIX, TOSHIY, TOSHINO,TOSHINO2, TOSHI1, TOSHI2
```

などはすべて異なった変数としてみなされます。

TOSHI(X) のＸは添え字としてみなされるのに対して TOSHIX のＸは，変数名の一部としてみなされてしまうからです。

いろいろ試してください。

```
>K↵

>S A(1)="ABC",A(2)="XXX",A(5)="ZZZ",A(100)="QQQ"↵
>W↵

A(1)="ABC"
A(2)="XXX"　　：今これだけの変数がセットされています
A(5)="ZZZ"
A(100)="QQQ"
>
```

添え字を変数で指定しましょう。

```
>S X=1,Y=1,Z=1↵

>W A(X)," ",A(Y)," ",A(Z)↵
ABC ABC ABC                          ：A(1)の値が3回表示されました
>
```

X, Y, Zがすべて1なので A(X), A(Y), A(Z) はすべて A(1) のことです。

もう一度トライしましょう。

```
>S SUB=100,NO=100,XX=20↵

>W A(SUB)," ",A(NO)," ",A(XX*5)↵
QQQ QQQ QQQ                          ：A(100)の値が3回表示されました
>
```

SUB=100, NO=100 なので, A(SUB), A(NO) が A(100) となるのは明白です。一方 XX=20 なので, XX*5 も 100 になります。したがって A(XX*5) も A(100) のことです。

もう少し複雑な例です。

```
>S A(4)=100,B=4↵

>W A(100)," ",A(A(4))," ",A(A(B))↵
QQQ QQQ QQQ                          ：またA(100)の値が3回表示されました
>
```

この例では，添え字に添え字付き変数が使われています。少し複雑なので詳しく解説しましょう。

原則は，MUMPS ではまず添え字に使われている変数から値を求めるということです。

A(A(4)) を求めるには，まず A(4) の値を求めます。A(4)＝100 なので，A(A(4)) は A(100) のことになります。

次に A(A(B)) は A(B) の値を求めようとします。しかしこれも添え字に変数が入っているので，その変数Bの値を第一に求めます。B＝4 ですから，A(B) は A(4) のことになります。そして A(4) の値は 100 ですから，A(B)＝100 です。したがって，A(A(B)) も A(100) のことになります。

結局最後の例でも，QQQ という値が3回表示されました。

- 変数には添え字を付けることができる
  例；TOSHI(1),SEI(10),A(50)
- 添え字の値が異なると別の変数として扱われる
  例；SEI(1),SEI(2),SEI(5)
- 添え字として変数を用いることができる
  例；SEI(X),SEI(NO),A(Z)
- VAR(SUB)とVARSUBは別の変数である
- X＝5,Y＝4のとき，A(X)とA(Y＋1)は同じ変数である

4,添え字付き変数を利用してデータを蓄える

添え字付き変数を利用すると多人数のデータを入力するルーチンを簡単に作ることができます。

list 5-3

```
MB003   ;YS,ｼｮｸｲﾝﾒｲﾎﾞ;1985.2.19;ﾃﾞｰﾀ ﾆｭｳﾘｮｸ(ｿｴｼﾞ ﾉ ﾘﾖｳ)
        ;
START   R !,"ﾊﾞﾝｺﾞｳ ? ",NO
        R !,"ﾅﾏｴ ..... : ",NAMAE(NO)
        R !,"ｾｲﾍﾞﾂ .... : ",SEI(NO)
        R !,"ﾈﾝﾚｲ ..... : ",TOSHI(NO)
        R !,"ｼｮｿﾞｸ･ﾌﾞｼｮ : ",BUSHO(NO)
        Q
```

このルーチンでは各人を適当な番号を付けて入力するようにしてあります。MUMPS の標準規格では添え字の数値として，0から 999999999 まで使用できるので，実用上無制限に番号を付けることができます。

このルーチンを入力して実行してください（ディスクへの登録も忘れずに）。

```
>D MB003↵                          ：ルーチンを起動します

ﾊﾞﾝｺﾞｳ ? 1↵
ﾅﾏｴ ..... : ｲﾜｻｷ ﾋﾛﾐ↵
ｾｲﾍﾞﾂ .... : F↵
ﾈﾝﾚｲ ..... : 26↵
ｼｮｿﾞｸ･ﾌﾞｼｮ : ｺｳﾎｳｼﾝﾌﾞ↵
>D MB003↵                          ：またルーチンを起動します

ﾊﾞﾝｺﾞｳ ? 2↵
```

```
ﾅﾏｴ ..... : ﾔﾏﾀﾞ ﾀﾛｳ↵
ｾｲﾍﾞﾂ .... : M↵
ﾈﾝﾚｲ ..... : 19↵
ｼｮｿﾞｸ･ﾌﾞｼｮ : ﾍﾝｼｭｳﾌﾞ↵
>
```

こうして何人ものデータを入力しておくことができます。期待通りに変数がセットされているかどうか確かめてください。

```
>W↵                              ：引数を持たないWRITE命令

BUSHO(1)="ｺｳﾎｳｼｼｭ"
BUSHO(2)="ﾍﾝｼｭｳﾌﾞ"
NAMAE(1)="ｲﾜｻｷ ﾋﾛﾐ"
NAMAE(2)="ﾔﾏﾀﾞ ﾀﾛｳ"              ：他にも変数が表示されるかもしれませんが，
NO=2                               それは今までにセットしてきた変数が
SEI(1)="F"                         そのまま残っているためです
SEI(2)="M"
TOSHI(1)="26"
TOSHI(2)="19"

>
```

5,入力したデータを書き出すルーチン

引数なしの WRITE 命令ではデータをスマートに表示できません。またいちいち変数を指定して WRITE 命令でデータを表示させるのも大変です。

次にデータを表示するルーチンを作ってみましょう。これも簡単なものです。ただし，このルーチンを入力する前に，データ入力ルーチンをディスクに登録してください(ZSAVE 命令を使うか，X　%US としてください)。さらに ZREMOVE 命令でワークエリアをきれいにしておくことを忘れないこと

list 5-4

```
MB004   ;YS,ｼｮｸｲﾝﾒｲﾎﾞ;1985.2.19;ﾃﾞｰﾀ ﾉ ﾋｮｳｼﾞ
        ;
START   R !,"ﾊﾞﾝｺﾞｳ ? ",NO
        W !,"ﾅﾏｴ ..... : ",NAMAE(NO)
        W !,"ｾｲﾍﾞﾂ .... : ",SEI(NO)
        W !,"ﾈﾝﾚｲ ..... : ",TOSHI(NO)
        W !,"ｼｮｿﾞｸ･ﾌﾞｼｮ : ",BUSHO(NO)
        Q
```

です。

番号を指定するとその人のデータが表示されます。

```
>D MB004↵

ﾊﾞﾝｺﾞｳ ? 2↵
ﾅﾏｴ ..... : ﾔﾏﾀﾞ ﾀﾛｳ
ｾｲﾍﾞﾂ .... : M
ﾈﾝﾚｲ ..... : 19
ｼｮｿﾞｸ･ﾌﾞｼｮ : ﾍﾝｼｭｳﾌﾞ
>
```

ここで，登録されていない番号を指定すると

```
>D MB004↵          登録されていない番号
ﾊﾞﾝｺﾞｳ ? 23↵
ﾅﾏｴ ..... :
<UNDEF> 定義されていない変数が参照されました
 1) +4^MB004:: W !,"ﾅﾏｴ ..... : ",NAMAE(NO)
 0) *XECUTE*^MB004:: D MB004
<UNDEF>+4^MB004:::4:0 W !,"ﾅﾏｴ ..... : ",NAMAE(NO)
```

というエラーが出てルーチンの実行が停止するはずです。これはルーチン MB004 の4行目，すなわち行 START+1 で，変数 NAMAE(NO) を WRITE しようとしたところ，この変数に値がセットされていなかったことを示しています。

- 定義されていない変数を参照しようとすると，
「<UNDEF> 定義されていない変数が参照されました」
というエラーが出てルーチンの実行が止まってしまう。

6,データをリストアップする

FOR命令

いちいち番号を指定してひとりだけのデータを表示するのももどかしくなります。全員のデータを整然とリストアップするルーチンがあると便利でしょう。次がそのルーチンです。

つづいて実行してみます。

list 5-5

```
MB005   ;YS,ｼｮｸｲﾝﾒｲﾎﾞ;1985.2.19;ﾃﾞｰﾀ ﾉ ﾘｽﾄ･ｱｯﾌﾟ
        ;
START   R !,"ｻｲｺﾞ ﾉ ﾊﾞﾝｺﾞｳ ﾊ ? ",NN
        F J=1:1:NN W !,J,?5,NAMAE(J),?20,SEI(J),?25,TOSHI(J),?30,BUSHO(J)
        W !,"*** ｵﾜﾘ ***"
        Q
```

```
>D MB005↵

ｻｲｺﾞ ﾉ ﾊﾞﾝｺﾞｳ ﾊ ? 5
1     ｲﾜｻｷ ﾋﾛﾐ          F    26    ｺｳﾎｳｼｽﾂ
2     ﾔﾏﾀﾞ ﾀﾛｳ          M    19    ﾍﾝｼｭｳﾌﾞ
3     ｼﾏ ﾖｼﾅﾘ           M    30    ｲﾑｼｽﾂ
4     ﾋﾗﾉ ｼﾞｭﾝｺ         F    27    ﾋｼｮｶ
5     ｷﾀﾑﾗ ｹﾝ           M    36    ｾｯｹｲﾌﾞ
*** ｵﾜﾘ ***
>
```

このルーチンには FOR 命令という新しい命令が入っています。FOR 命令は一言で表すと，作業を繰り返すための命令です。

FOR 命令のもっとも頻繁に使われるのは

```
F J=A:B:C 他の命令
```

という形です。これは

①Ｊという変数にＡという値をセットして他の命令を１回実行

②Ｊの値にＢを加えてまた他の命令を１回実行

③さらにＪの値にＢを加えてまた他の命令を１回実行

④こうしてＪの値がＣになるまで他の命令を繰り返す

ということです。

「変数ＪをＡからＢずつ増やしてＣになるまで他の命令を繰り返す」と読むことができます。

たとえば

```
F J=1:1:10 W J
```

とすれば，変数Ｊの値を，始めに１，それを１ずつ増やして，10になるまでＷ　Ｊを繰り返して実行する（Ｊを１から１ずつ10まで変化させてＷＪを繰り返す）ことになります。したがってこれは

```
S J=1 W J
S J=2 W J
S J=3 W J
   :
   :
S J=10 W J
```

と，SET 命令と WRITE 命令を10回繰り返すことと同じ意味です。そして，モニターテレビ画面には

```
12345678910
```

と表示されます。また

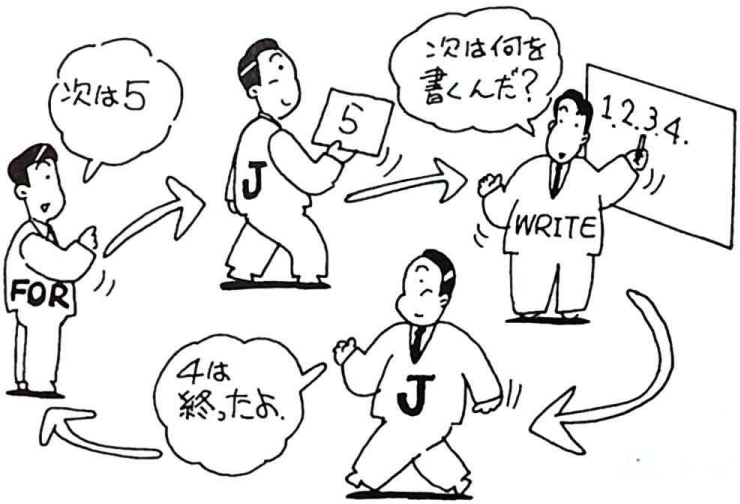
```
F K=4:2:12 W "ABC"
```

とすれば，変数Kが4, 6, 8, 10, 12と変化するごとに W "ABC" が実行されるので，計5回 ABC が表示されます。

```
ABCABCABCABCABC
```

FOR 命令に用いる変数の名前は J, K に限りません。添え字付き変数であってもかまいません（Chapter 6 参照，グローバル変数は用いることができません）。

いろいろと確かめてみましょう。



```
>F K=1:1:10 W K," "↵                    ：W K," "を10回繰り返します
1 2 3 4 5 6 7 8 9 10
>F A=2:1:7 W A,"**"↵                   ：W A,"＊＊" を６回繰り返します
2**3**4**5**6**7**
>F X(1)=2:2:10 W "/",X(1)↵             ：W "/",X(1)を５回繰り返します
/2/4/6/8/10
>F ZZ=3:3:12 W !,"I Love You.  ZZ=",ZZ

I Love You.  ZZ=3
I Love You.  ZZ=6                      ：W !,"I Love You. ZZ=",ZZを
I Love You.  ZZ=9                        ４回繰り返しました
I Love You.  ZZ=12
>
```

FOR 命令の右側の命令は，何の命令でもかまいません。

READ 命令を繰り返してみましょう。

```
>F X=1:1:4 R !,"DATA ? ",Y(X)↵
DATA ? 35↵          ：35という値はY(1)にセットされます
DATA ? 10↵          ：10という値はY(2)にセットされます
DATA ? 24↵          ：24という値はY(3)にセットされます
DATA ? ABCD↵        ：ABCDという値はY(4)にセットされます
>
```

READ 命令で入力したデータを書き出してみましょう。

```
>F J=1:1:4 W Y(J),","↵
35,10,24,ABCD,
>
```

入力のときには Y(X) だったのに，表示するときには Y(J) となっていることに抵抗を感じるかもしれません。しかし添え字付き変数は，添え字を決める変数の名前ではなく，その添え字の値によって区別されるということを思い出してください。

XもJも４ならば，Y(X) も Y(J) も Y(4) とみなされます。

FOR 命令の『＝』の右側の繰り返しを指定する数値にも変数を用いることができます。

```
>S N=10↵

>F KX=1:1:N W KX," "↵
```

```
1  2  3  4  5  6  7  8  9  10
>

>S N=12↵

>F KX=1:1:N W KX,"  "↵
1  2  3  4  5  6  7  8  9  10  11  12
>
```

まったく同じ命令でも出力した結果が異なります

このふたつの例では KX という変数の最終値をNという変数で指定しています。

さて最初の例題に戻りましょう。例題のリストアップルーチンMB005でも繰り返す回数を変数で指定しています。

行 START とその次の行を見てください。READ 命令で入力した変数 NN を FOR 命令の繰り返しを指定する値として使っています。

```
START    R !,"ｻｲｺﾞ ﾉ ﾊﾞﾝｺﾞｳ ﾊ ? ",NN            :繰り返す回数の入力
         F J=1:1:NN W !,J,?5,NAMAE(J),?20,SEI(J),?25,TOSHI(J),?30,BUSHO(J)
```

NN回WRITEします

- FOR命令は同じ行のその右側にある命令を繰り返す
- F J＝A:B:C は変数Jの初期値A，増分B，最終値Cで変化させて，右側の命令を繰り返すことを指定する

7,データを表示する位置を制御する

前節の例題の行 START＋1 の WRITE 命令は

```
W !,J,?5,NAMAE(J),?20,SEI(J),?25,TOSHI(J),?30,BUSHO(J)
```

となっています。この中に ?5, ?20, ?25, ?30 という引数（下線が引いてある）が使われています。これは左端から何文字あけるかということを指定する引数です。たとえば

```
W ?10,X
```

とすると，画面のその行にすでに何文字表示されていても，左端から10文字あけてからXの値を表示します。

試してみましょう。

```
>W !,"ABCD",?10,"XYZ"↵

ABCD      XYZ
>
```

ABCD と XYZ の間には，空白が6文字分あります。XYZ は左端から10文字あけて11文字目から表示されています。

もう一度トライしましょう。

```
>W !,"ABCDEF",?10,"XYZ"↵

ABCDEF    XYZ
>
```

ABCDEF と XYZ の間には，空白が4文字あります。やはり XYZ は左端から10文字あけて11文字から表示されています。

```
>W !,"12345678",?10,"XYZ"↵

12345678  XYZ
>
```

前の2例と XYZ の位置が同じであることを確認してください。

ところで，W ?10 としたとき，もしすでに10文字以上表示されていた場合はどうなるでしょうか。試してください。

```
>W !,"1234567890123456",?10,"XYZ"↵

1234567890123456XYZ                    ：XYZが右へズレていることに注意！
>
```

XYZ は前の例より右に表示されています。

このように，すでに指定された文字数以上の文字が表示された場合には，左に戻って表示することはせず，右の方へ続けて表示されます。

ところで例題には無関係ですが

```
W !,?10
```

のように『！』記号と『？』記号が続いているときに限り，コンマ『，』を省略して

```
W !?10
```

と表すことができます。

- W　?nは次のデータを表示する前に左端からn文字空けることを指定する
- W！，?nはW　!?nと省略できる

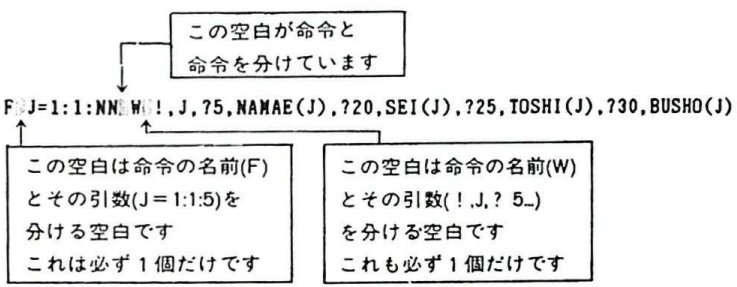
8,命令を複数並べる

リスト5-5のルーチンについてはもうひとつ解説しておくことがあります。このルーチンの行 START+1 には命令がふたつあります。

```
F J=1:1:NN W !,J,?5,NAMAE(J),?20,SEI(J),?25,TOSHI(J),?30,BUSHO(J)
```

FOR命令　WRITE命令

今までは命令を1行にひとつしか書いてきませんでしたが，実は1行にふたつ以上の命令を並べることもできます。この場合は，命令と命令の間に1個以上の空白を入れなくてはなりません。この FOR 命令とその右にある命令（ここでは WRITE 命令）の間には空白が1個あります。



FOR 命令以外の命令も複数並べて書くことができます。

```
>S X=9999 W X↵
9999
>
```

これは，命令をひとつずつ書いた場合と同じことです。

```
>S X=9999↵
```

```
>W X↵
9999
>
```

ところで，FOR 命令だけは特別です。FOR 命令とその右の繰り返すべき命令（上の例では WRITE 命令）を別々の行にしてはいけません。それは，FOR 命令が繰り返しを制御するのは，同じ行のそれも FOR 命令より右側にある命令だけだからです。

```
>F J=1:1:100↵
                    ：何も起こりません
>W J↵
100                 ：1回しか表示しません
>
```

FOR 命令だけでは何も起こりません。いや，FOR 命令だけが100回空回りしたのです。そして，別にタイプした WRITE 命令は 1 回実行されただけです。

FOR 命令や，次の節で解説する IF 命令のように，他の命令の実行を制御する命令は，1 行に並べて書いたときとそうでないときでは意味が異なるので注意してください。

- 命令は 1 行に並べて書くことができる
- 命令が 1 行に並ぶときには，命令と命令の間に空白文字を 1 個以上いれる
- FOR命令，IF命令などの他の命令の実行を制御する命令を用いるときは，命令を 1 行に並べたときと別の行に書いたときとで，意味が変わる
- 他の命令の実行を制御することのない命令は，1 行で書いても，別々の行に書いても変わらない

9，女性のみのリストアップ

IF命令

前節までで解説した全データをリストアップするルーチン MB005 に，ほんの少し手を加えるだけで，ある判断基準に合う人のデータだけをリストアップするように変更することができます。

これは女性のみのデータをリストアップするルーチンです。

list 5-6

```
MB006   ;YS,ｼｮｸｲﾝﾒｲﾎﾞ;1985.2.19;ｼﾞｮｾｲ ﾉﾐ ﾉ ﾃﾞｰﾀ ﾉ ﾘｽﾄ･ｱｯﾌﾟ
        ;
START   R !,"ｻｲｺﾞ ﾉ ﾊﾞﾝｺﾞｳ ﾊ ? ",NN
        F J=1:1:NN I SEI(J)="F" W !,J,?5,NAMAE(J),?20,SEI(J),?25,TOSHI(J),?30,BU
    -->SHO(J)
        W !,"*** ｵﾜﾘ ***"
        Q
```

これは前の行の続きで表示幅の都合で曲がって
ここに表示されているという印です。
ZPRINTしても実際には表れません

入力して実行してみましょう（「ルーチン修正のヒント」参照）

```
>D MB006↵

ｻｲｺﾞ ﾉ ﾊﾞﾝｺﾞｳ ﾊ ? 5↵
1     ｲﾜｻｷ ﾋﾛﾐ          F    26    ｺｳﾎｳｼｼﾂ
4     ﾋﾗﾉ ｼﾞｭﾝｺ          F    27    ﾋｼｮｶ
*** ｵﾜﾘ ***
>
```

確かに女性だけがリストアップされています。これは行 START+1 の FOR 命令のすぐ右側にある IF 命令という新しい命令の働きです。

IF 命令は与えられた条件を判断して，その右側にある他の命令の実行を禁止したり許可したりします。IF 命令の条件が正しければ右側にある命令には何の影響も与えません。しかし条件が誤っている場合には右側の命令の実行を禁止します。たとえば

```
I X=1 W "X ﾊ 1 ﾃﾞｱﾙ"
```

とすると，変数Xの値が1の場合には「Xハ1デアル」と表示しますが，そうでない場合には何も表示しません。

同様に例題のルーチン MB006 では，ある人のデータを画面に表示するか否かを行 START+1 の IF 命令（I SEI(J)="F" 部分）で判断しています。

```
F J=1:1:NN I SEI(J)="F" W !,J,?5,NAMAE(J),?20,SEI(J),?25,...  （省略）
```

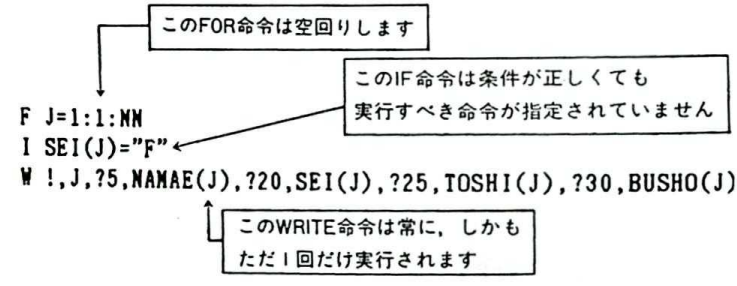
この行の左端には FOR 命令がありますから，変数Jの値を1からひとつずつ増やして変数 NN に等しくなるまでこの IF 命令を含めた右側の命令を繰り返して実行します。

そして IF 命令は FOR 命令で繰り返されるごとに，変数 SEI(J) の値（すなわち J 番目の人の性別）が 'F' であるかどうかを判断します。もしそれが正しいときには，さらにその右側にある WRITE 命令を実行します。

この結果，女性のみのデータリストが表示されます。

このルーチンの行 START+1 には 1 行に命令が 3 つあります。

FOR 命令と IF 命令と WRITE 命令です。これらの命令はそれぞれ 1 個の空白で区切られています。そしてこれらの命令は，別々の行にしてしまっては意味がなくなることに注意してください。



IF 命令を理解するためにもう少し例を試してみましょう。

```
YSTST1  ;YS,SAMPLE;1986.1.15;IF ﾒｲﾚｲ ﾉ ﾚｲ
        ;
READX   R !,"X ? ",X
HANDAN1 I X<10 W ?10,"X ﾊ 10 ﾖﾘ ﾁｲｻｲ"
HANDAN2 I X=10 W ?10,"X ﾊ 10 ﾃﾞｱﾙ"
HANDAN3 I X>10 W ?10,"X ﾊ 10 ﾖﾘ ｵｵｷｲ"
        W !
        Q
```

このルーチンはXに入力されたデータを10と比較して

10より小さい場合はXハ10ヨリチイサイ

10に等しい場合はXハ10デアル

10より大きい場合はXハ10ヨリオオキイ

と表示します，試してみましょう。

```
>D YSTST1↵

X ? 5↵    X ﾊ 10 ﾖﾘ ﾁｲｻｲ
```

```
>D YSTST1↵

X ? 23↵   X ハ 10 ヨリ オオキイ

>D YSTST1↵

X ? 10↵   X ハ 10 デアル

>
```

いろいろな値を入力してください。ここで

「＝」記号は，左辺と右辺が等しい
「＞」記号は，左辺は右辺より大きい
「＜」記号は，左辺は右辺より小さい

という判断に使われています。

IF 命令の「＝」記号は SET 命令の「＝」記号とは意味が違います。IF 命令では等しいか否かの判断をするという意味であり，SET 命令は右辺の計算結果を左辺の変数にセットするという意味です。

◎ルーチン修正のヒント

女性だけのデータをリストアップするルーチン MB006 は，はじめからタイプする必要はありません。全データをリストアップするルーチン MB005 を修正するだけでできあがります。

```
>ZL MB005↵              ：MB005をディスクから取り出します
>X ^Z↵                  ：エディタを起動します
編集行: MB005↵          ：1行目を修正します
MB005     ;YS,ｼｮｸｲﾝﾒｲﾎﾞ;1985.2.19;ﾃﾞｰﾀ ﾉ ﾘｽﾄ･ｱｯﾌﾟ
  変更したい文字: 005↵  変更後の文字: 006↵  変更したい文字: ﾃﾞｰﾀ↵
  変更後の文字: ｼﾞｮｾｲ ﾉﾐ ﾉ ﾃﾞｰﾀ↵  変更したい文字: ↵
MB006     ;YS,ｼｮｸｲﾝﾒｲﾎﾞ;1985.2.19;ｼﾞｮｾｲ ﾉﾐ ﾉ ﾃﾞｰﾀ ﾉ ﾘｽﾄ･ｱｯﾌﾟ
編集行: MB005 // START+1↵      ：行START 1を修正します
F J=1:1:NN W !,J,?5,NAMAE(J),?20,SEI(J),?25,TOSHI(J),?30,BUSHO(J)
  変更したい文字: W↵  変更後の文字: I SEI(J)="F" W↵  変更したい文字: ↵
F J=1:1:NN I SEI(J)="F" W !,J,?5,NAMAE(J),?20,SEI(J),?25,TOSHI(J),?30,SHO
```

行ラベル修正[4]

IF命令の追加

```
.........  ZOKU(J)
編集行: START+1 // ␣⏎           :エディタを終了します
>X ^%US⏎                        :ディスクに登録します
      [ルーチン名が変わりました]
[MB006] はディスクに格納されました。

>ZP⏎                            :ルーチンの内容を表示します
MB006    ;YS,ショクインメイボ;1985.2.19;ジョセイ ノミ ノ データ ノ リスト・アップ
         ;
START    R !,"サイコ ノ バンゴウ ハ ? ",NN
         F J=1:1:NN I SEI(J)="F" W !,J,?5,NAMAE(J),?20,SEI(J),?25,TOSHI(
J),?30,BUSHO(J)
         W !,"*** オワリ ***"
         Q                          [IF命令が加わりました]

>
```

混み入っていますが，ユーザーがタイプしたところには下線が引いてあるので参考にしてください。慣れれば最初からタイプしなおすより，ルーチンを修正するほうが速くなります。

- IF命令は条件を判断して，それが正しければ(「真」)その右側にある命令を実行し，正しくなければ(「偽」)右側の命令の実行を禁止する
- IF命令の形式は，I 条件　次の命令
- 「=」記号は，SET命令とIF命令では意味が異なる。SET命令の「=」は，右辺の式の計算結果を左辺の変数に割り付けることを表し，IF命令の「=」は，右辺と左辺の値が等しいか否かを判断することを表す

10，いろいろな条件のリスト

ルーチン MB006 を改造していろいろな条件のリストを作るようにしましょう。IF 命令の条件式のみを変更すればこと足ります。

■ notes

4）エディタを用いて行ラベルを修正することができます。この例では，行ラベル'MB005'を'MB006'に修正しています。
　エディタでは，行ラベルとその行の先頭の命令との間の TAB は1個の空白（スペースバー）で代用することができます。

① 男性のみのリスト

```
I SEI(J)="F"
```

を次のように変更します。

```
I SEI(J)="M"
```

list 5-7

```
MB007   ;YS,ｼｮｸｲﾝﾒｲﾎﾞ;1985.2.19;ﾀﾞﾝｾｲ ﾉﾐ ﾉ ﾃﾞｰﾀ ﾉ ﾘｽﾄ･ｱｯﾌﾟ
        ;
START   R !,"ｻｲｺﾞ ﾉ ﾊﾞﾝｺﾞｳ ﾊ ? ",NN
        F J=1:1:NN I SEI(J)="M" W !,J,?5,NAMAE(J),?20,SEI(J),?25,TOSHI(J),?30,BU
     -->SHO(J)
        W !,"*** ｵﾜﾘ ***"
        Q
```

```
>D MB007↵

ｻｲｺﾞ ﾉ ﾊﾞﾝｺﾞｳ ﾊ ? 5↵
2     ﾔﾏﾀﾞ ﾀﾛｳ          M    19    ﾍﾝｼｭｳﾌﾞ
3     ｼﾏ ﾖｼﾅﾘ           M    30    ｲﾑｼﾂ
5     ｷﾀﾑﾗ ｹﾝ           M    36    ｾｯｹｲﾌﾞ
*** ｵﾜﾘ ***
>
```

② 30歳未満の人のリスト

条件は次のようにすればよいことはすぐに理解できるでしょう。

```
I TOSHI(J)<30
```

list 5-8

```
MB008   ;YS,ｼｮｸｲﾝﾒｲﾎﾞ;1985.2.19;30 ｻｲ ﾐﾏﾝ ﾉ ﾃﾞｰﾀ ﾉ ﾘｽﾄ･ｱｯﾌﾟ
        ;
START   R !,"ｻｲｺﾞ ﾉ ﾊﾞﾝｺﾞｳ ﾊ ? ",NN
        F J=1:1:NN I TOSHI(J)<30 W !,↓,?5,NAMAE(J),?20,SEI(J),?25,TOSHI(J),?30,B
     -->USHO(J)
        W !,"*** ｵﾜﾘ ***"
        Q
```

```
>D MB008↵

ｻｲｺﾞ ﾉ ﾊﾞﾝｺﾞｳ ﾊ ? 5↵
1     ｲﾜｻｷ ﾋﾛﾐ          F    26    ｺｳﾎｳｼﾂ
2     ﾔﾏﾀﾞ ﾀﾛｳ          M    19    ﾍﾝｼｭｳﾌﾞ
4     ﾋﾗﾉ ｼﾞｭﾝｺ          F    27    ﾋｼｮｶ
*** ｵﾜﾘ ***
>
```

③ 指定した年齢未満の人のリスト

任意の年齢を指定するとその年齢未満の人のデータのリストを表示するルーチンです。どこが変更されていますか？

list 5-9

```
MB009   ;YS,ｼｮｸｲﾝﾒｲﾎﾞ;1985.2.19;ﾆﾝｲ ﾉ ﾈﾝﾚｲ ﾐﾏﾝ ﾉ ﾃﾞｰﾀ ﾉ ﾘｽﾄ･ｱｯﾌﾟ
        ;
START   R !,"ｻｲｺﾞ ﾉ ﾊﾞﾝｺﾞｳ ﾊ ? ",NN
        R !,"ﾅﾝｻｲ ﾐﾏﾝ ﾉ ﾋﾄ ﾉ ﾃﾞｰﾀ ﾖ ﾘｽﾄ ｼﾏｽｶ ? ",TOSHI
        F J=1:1:NN I TOSHI(J)<TOSHI W !,J,?5,NAMAE(J),?20,SEI(J),?25,TOSHI(J),?3
      -->0,BUSHO(J)
        W !,"*** ｵﾜﾘ ***"
        Q
```

```
>D MB009↵

 ｻｲｺﾞ ﾉ ﾊﾞﾝｺﾞｳ ﾊ ? 5↵
 ﾅﾝｻｲ ﾐﾏﾝ ﾉ ﾋﾄ ﾉ ﾃﾞｰﾀ ﾖ ﾘｽﾄ ｼﾏｽｶ ? 35↵
 1      ｲﾜｻｷ ﾋﾛﾐ          F    26    ｺｳﾎｳｼｯ
 2      ﾔﾏﾀﾞ ﾀﾛｳ          M    19    ﾍﾝｼｭｳﾌﾞ
 3      ｼﾏ ﾖｼﾅﾘ           M    30    ｲﾑｼｯ
 4      ﾋﾗﾉ ｼﾞｭﾝｺ         F    27    ﾋｼｮｶ
*** ｵﾜﾘ ***
>
```

このルーチンで TOSHI と TOSHI(J) が異なる変数であることに注意してください。

11, 名前による検索

データの集まりの中からある特定の条件に当てはまるデータを探し出す作業のことを「検索」と呼びます。

たとえばある名前の人のデータを探すことを「名前で検索する」といいま

list 5-10

```
MB010   ;YS,ｼｮｸｲﾝﾒｲﾎﾞ;1985.2.19;ﾅﾏｴ ｺ ﾖﾙ ｹﾝｻｸ
        ;
START   R !,"ｻｲｺﾞ ﾉ ﾊﾞﾝｺﾞｳ ﾊ ? ",NN
        R !,"ﾅﾏｴ ﾊ ? ",NAMAE
        F J=1:1:NN I NAMAE(J)=NAMAE W !,J,?5,NAMAE(J),?20,SEI(J),?25,TOSHI(J),?3
      -->0,BUSHO(J)
        W !,"*** ｵﾜﾘ ***"
        Q
```

す。list5-10 に名前で検索するルーチンを示します。前節のリストアップのルーチンとほとんど同じものです。IF 命令の条件式を NAMAE(J)=NAMAE にしただけです。ここでも変数 NAMAE と NAMAE(J) が異なることに注意してください。

```
>D MB010↵

ｻｲｺﾞ ﾉ ﾊﾞﾝｺﾞｳ ﾊ ? 5↵
ﾅﾏｴ ﾊ ? ﾔﾏﾀﾞ ﾀﾛｳ↵
2     ﾔﾏﾀﾞ ﾀﾛｳ          M    19    ﾍﾝｼｭｳﾌﾞ
*** ｵﾜﾘ ***
>
```

12, 名前の一部で検索する

ところで，名前全体で検索するよりは，姓のみで検索することの方が便利なことが多いと思います。MUMPS ではこのような機能を『[』という記号を用いて簡単に実現することができます。

list 5-11

```
MB011   ;YS,ｼｮｸｲﾝﾒｲﾎﾞ;1985.2.19;ﾅﾏｴ ｺ ﾖﾙ ｹﾝｻｸ
        ;
START   R !,"ｻｲｺﾞ ﾉ ﾊﾞﾝｺﾞｳ ﾊ ? ",NN
        R !,"ﾅﾏｴ ﾊ ? ",NAMAE
        F J=1:1:NN I NAMAE(J)[NAMAE W !,J,?5,NAMAE(J),?20,SEI(J),?25,TOSHI(J),?3
     -->0,BUSHO(J)
        W !,"*** ｵﾜﾘ ***"
        Q
```

```
I NAMAE(J)=NAMAE ①
I NAMAE(J)[NAMAE ②
```

つまり①の代わりに②とします。『[』記号は「包含演算子」と呼ばれ，左辺のデータが右辺のデータを含んでいるか否かを判断します。たとえば

```
"ｲﾜｻｷ ﾋﾛﾐ"["ｲﾜｻｷ"          :'イワサキヒロミ'は'イワサキ'を含んでいる
"ﾋﾗﾉ ｼﾞｭﾝｺ"["ﾋﾗﾉ"          :'ヒラノジュンコ'は'ヒラノ'を含んでいる
```

という式は正しい（「真」であり）

```
"ｲﾜｻｷ ﾋﾛﾐ"["ﾏｯﾀﾞ"
"ｲﾜｻｷ ﾋﾛﾐ"["ｲｻｷ"          :'イワサキヒロミ'は'イサキ'を含んでいない
"ﾋﾗﾉ ｼﾞｭﾝｺ"["ﾋﾗﾀ"          :'ヒラノジュンコ'は'ヒラタ'や'ヒノ'を含んでいない
"ﾄﾗﾉ ｼﾞｭﾝｺ"["ﾋﾉ"
```

などは正しくない（「偽」）と判断されます。

この『[』（包含演算子）を用いてルーチンを改造します。

```
>D MB011

ｻｲｺﾞ ﾉ ﾊﾞﾝｺﾞｳ ﾊ ? 5
ﾅﾏｴ ﾊ ? ﾔﾏﾀﾞ
2     ﾔﾏﾀﾞ ﾀﾛｳ          M    19    ﾍﾝｼｭｳﾌﾞ
*** ｵﾜﾘ ***
>D MB011

ｻｲｺﾞ ﾉ ﾊﾞﾝｺﾞｳ ﾊ ? 5
ﾅﾏｴ ﾊ ? ｲﾜｻｷ
1     ｲﾜｻｷ ﾋﾛﾐ          F    26    ｺｳﾎｳｼｼﾂ
*** ｵﾜﾘ ***
>
```

『[』演算子は右辺のデータが左辺のデータのどこに含まれているかということまでは判断していないので，どこかに含まれてさえいれば「真」として判断してしまいます。したがって

```
"ｲﾜｻｷ ﾋﾛﾐ"["ﾋﾛﾐ"
"ｲﾜｻｷ ﾋﾛﾐ"["ｻｷ"
"ﾋﾗﾉ ｼﾞｭﾝｺ"["ｼ"
"ｼﾏ ﾖｼﾅﾘ"["ｼ"
```

はすべて「真」になります。試してください

```
>D MB011

ｻｲｺﾞ ﾉ ﾊﾞﾝｺﾞｳ ﾊ ? 5
ﾅﾏｴ ﾊ ? ｼ
3     ｼﾏ ﾖｼﾅﾘ            M    30    ｲﾑｼｽ
4     ﾋﾗﾉ ｼﾞｭﾝｺ          F    27    ﾋｼｮｶ
*** ｵﾜﾘ ***
>
```

名前のどこかに 'シ' の入っている人のデータが表示されました。

- 包含演算子『[』は，その左辺のデータが右辺のデータを含んでいるか否かを判断する

Chapter 6

データを永久保存する

前章で，職員名簿のデータを蓄え，それを用いて種々の作業をするルーチン群を作ってきましたが，これらのルーチンの大きな問題点は，データを永久保存できず，コンピュータの電源を切るとデータが消えてしまうということです。

この章では，「グローバル変数」と呼ばれる MUMPS に独特な変数を用いることを解説します。これで，データをディスク上に半永久的に保存することができるようになります。

1,グローバル変数

今までは職員名簿のデータを処理するルーチンをいくつも作ってきました。そのデータはすべてコンピュータ内部のワークエリアに入っています。しかし，これらはコンピュータの電源を切ってしまうとすべて消えてしまうので，後日また使うことはできません。

一方，MUMPS にはグローバル変数という独特な変数があり，これを用いるとデータをディスク上に蓄えることができます。ディスク上のデータは，電源を切っても消えないので，これは半永久的に保存することができます。

加えて，ディスクには容量がきわめて大きいという利点もあります。ワークエリア上にはたかだか100人程度のデータしか入らないかもしれません。ところがフロッピーディスクには何千人ものデータが入り，ハードディスクを用いると何万人分のデータを蓄えることもできます。

グローバル変数といっても難しいものではありません。ただ今まで利用してきた変数（これらをグローバル変数に対してローカル変数と呼びます）の名前の前に『^』（アッパーキャレットまたはサーカムフレクス）記号を付けるだけのことです。

たとえば

```
A, B12, XOX, ANSWER, NAMAE(1), SEI(20), TOSHI(J)
```

といったローカル変数はワークエリア上に蓄えられ

```
^A, ^B12, ^XOX, ^ANSWER, ^NAMAE(1), ^SEI(20), ^TOSHI(J)
```

はグローバル変数としてディスク上に格納されます。

グローバル変数とローカル変数は『^』記号の有無だけで区別されます。ルーチンを作る上でもほとんど違いはありません。グローバル変数の名前として使用できる文字も，先頭の『^』記号を除くと

①アルファベットの大文字

②数字

で，第1字には数字は使えません。『^』記号を除いて，名前の最大文字数は8文字です。すべてローカル変数と同じ規則です。

ただし，グローバル変数は READ 命令と，FOR 命令の引数の中では原則として使用できないという制限があります。

```
R !,"ナマエ ? ",^NAMAE
F ^J=1:1:5 W "MUMPS"
```

という使い方はできません。

ローカル変数とグローバル変数を比較してみましょう。

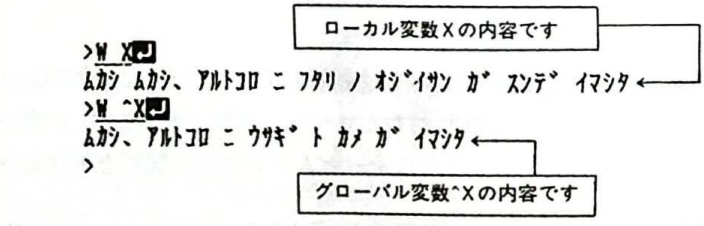
```
>S X="ムカシ ムカシ、アルトコロ ニ フタリ ノ オジイサン ガ スンデ イマシタ"↵
```

データをコンピュータ内部のワークエリア上に記憶させました。

```
>S ^X="ムカシ、アルトコロ ニ ウサギ ト カメ ガ イマシタ"↵
```

データをディスク上に記憶させました。

グローバル変数をセットしたときにディスク装置のランプが点灯し，カチッカチッとかブンブンといった音が聞こえなかったでしょうか？　さて，セットした内容を表示してみましょう。



Xと^Xは別々の変数として扱われています。

もう少し，グローバル変数を SET したり，WRITE してみましょう。ローカル変数と同じように扱えることを覚えてください。

```
>S ^X(1)="ABC"↵                    ：グローバル変数^X(1)をセットします

>S ^X(2)="QQQ"↵                    ：グローバル変数^X(2)をセットします

>S ^X(10)="ZZZ"↵

>W ^X(1)↵                          ：グローバル変数^X(1)を表示します
ABC
>W ^X(2),"  ",^X(10)↵
QQQ  ZZZ
>I ^X(2)="QQQ" W "YES"↵            ：IF 命令の中ではグローバル変数も使用できます
YES
>S X(1)="XYZ"↵

>S X(2)="RRRR"↵

>W X(1),"  ",X(2),"  ",^X(1),"  ",^X(2)↵
XYZ  RRRR  ABC  QQQ                ：ローカル変数 X(1) とグローバル変数^X(1)に
>                                    別のデータが入っていることに注意
```

ここでいったん MUMPS の仕事を終了してください。

```
>H↵

EXIT

A>          ：MS－DOSに戻りました
```

さらに再度 MUMPS を起動してください。

```
A>MSM -D B:DATABASE.MSM↵
```

その上でさきほどセットしたグローバル変数を WRITE します。

```
>W ^X(1),"  ",^X(2),"  ",^X(10)↵
ABC  QQQ  ZZZ                      ：グローバル変数の内容が表示されました
>
```

MUMPS はグローバル変数を忘れていませんでした。

ローカル変数はどうでしょう。

```
>W X(1)↵

<UNDEF> 定義されていない変数が参照されました
 0) *XECUTE*:: W X(1)                    :エラーメッセージ
<UNDEF>*XECUTE*:::4:0 W X(1)
>
```

ローカル変数を表示しようとしたら〈UNDEF〉というエラーメッセージが表示されたことが分かったでしょうか？　ローカル変数は MUMPS をいったん終了した時点ですべて消去されてしまったのです。

- グローバル変数を用いるとデータを半永久的に保存できるようになる
- グローバル変数は，名前の先頭に「^」記号を付ける
- グローバル変数の，「^」記号を除いた名前には，アルファベットの大文字と数字を使うことができる。ただし，数字は第1文字には使用できない
- 「^」記号の付いていない従来使用してきた変数をローカル変数という
- グローバル変数は，原則として READ 命令と FOR 命令の引数の中では使用できない

2，グローバル変数を用いたデータ入力ルーチン

前章のデータ入力ルーチンMB003を作りなおしてみましょう（リスト 6-1）。前節で述べたように READ 命令ではグローバル変数は使用できないので，READ 命令で入力するデータはリスト6-1のようにいったんローカル変数に入れてから改めてグローバル変数にセットします。

ここで，NAMAE(NO)，BUSHO(NO) といった変数を ^NAMAE(NO)，^BUSHO(NO) といったグローバル変数にせず，^MBNAMAE(NO)，^MBBUSHO(NO) という変数にしたのは理由があります。

ディスクには普通何種類ものデータを集めて登録しています。いまここでは「職員名簿」を処理していますが，同じディスクに「同好会会員名簿」や「取引先名簿」があるかもしれません。そのときに ^NAMAE，^TOSHI といっ

list 6-1

```
MBO12   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1985.2.20;ﾃﾞｰﾀ ﾆｭｳﾘｮｸ (ｸﾞﾛｰﾊﾞﾙ ﾍﾝｽｳ ﾉ ﾘﾖｳ)
        ;
HAJIME  R !,"ﾊﾞﾝｺﾞｳ : ",NO
        R !,"ﾅﾏｴ ..... : ",NAMAE
        R !,"ｾｲﾍﾞﾂ .... : ",SEI
        R !,"ﾈﾝﾚｲ ..... : ",TOSHI
        R !,"ｼｮｿﾞｸ･ﾌﾞｼｮ : ",BUSHO
KAKUNOU S ^MBNAMAE(NO)=NAMAE          ：MBはMBO12の最初の２文字
        S ^MBSEI(NO)=SEI
        S ^MBTOSHI(NO)=TOSHI
        S ^MBBUSHO(NO)=BUSHO
        Q
```

た誰でも使いそうな名前を使ってしまうと，他の目的のデータと混ざりあって混乱してしまいます。そこで，頭文字２文字をルーチンの名前と同じに統一しておくと，このような事態は防げます。

すでに登録されているグローバル変数の名前を知りたければ

```
>D ^%GD↵
```

としてください。グローバル変数名が表示されます。'GD' は Global Directry（グローバル登録台帳という意味）のことです。

ちなみに，ルーチン名を調べるときは

```
>D ^%RD↵
```

だったことを覚えていますか？（Chapter 3-11 参照）

- グローバル変数の頭文字２文字はルーチン名の頭文字２文字と統一しておくと他の目的のグローバル変数と混乱を起こさないので都合がよい。
- 登録されているグローバル変数名を調べるには D ^%GD とする。

3,添え字を複数つける

前節の例では，名前，性別，年齢，電話番号という各データ項目を

```
^MBNAMAE,  ^MBSEI,  ^MBTOSHI,  ^MBBUSHO
```

という４つのグローバル変数に別々に格納しています。この方法は直観的で分

かりやすいのですが，実用的には問題があります。

データ項目がこのように数少なく，登録されているデータも少ないときにはよいのですが，データ項目が数十から数百にわたり，登録人数も何百人から何万人にもなるときに問題になります。

全項目を別々の変数に入れておくと，数十数百ものグローバル変数を管理しなくてはなりません。これは現実的には大変困難です。登録されている各個人のデータをそろえることさえ手間がかかります。

そこで，普通は上記のようなグローバル変数の構成は採用せず，たとえば

```
名前……^MBM(NO,0)
性別……^MBM(NO,1)
年齢……^MBM(NO,2)
所属部署……^MBM(NO,3)
```

というように，ひとつのグローバル変数にすべての情報を入れてしまいます。

この例では，変数 NO が，各個人の登録番号で，0，1，2，3が各個人ごとのデータ項目を区別する添え字になります。添え字がコンマ『，』で区切ってふたつずつあることに注意してください。

MUMPS ではローカル変数，グローバル変数のどちらも，添え字をいくつも付けることができます。たとえば，上記のグローバル変数では

```
^MBM(12,0)……登録番号 12 の人の名前
^MBM(343,2)……登録番号 343 の人の年齢
^MBM(9877,1)……登録番号 9877 の人の性別
^MBM(19507,2)……登録番号 19507 の人の年齢
```

などとすることができます。

別の例をあげましょう。

```
^KAKEIBO(860221,5,1)……1986 年 2 月 21 日の 5 番目に買った品物の名前
^KAKEIBO(860221,5,2)……1986 年 2 月 21 日の 5 番目に買った品物の代金
^KAKEIBO(860221,5,3)……1986 年 2 月 21 日の 5 番目の支出の区分(例：衣料費)
^KAKEIBO(860303,2,1)……1986 年 3 月 3 日の 2 番目に買った品物の名前
^KAKEIBO(860416,10,2)…1986年 4 月 16 日の 10番目に買った品物の代金

^BUNKEN(34,0)……文献番号 34 番の文献名
^BUNKEN(56,1,3)……文献番号 56 番の項目 1(著者) の 3 番目の著者名
^BUNKEN(73,8,2)……文献番号 73 番の項目 8(キーワード)の 2 番目のキーワード
```

^SEITO(8,403,9,0)……登録番号 8 の学校の 403 組の出席番号 9 番の生徒の名前
^SEITO(8,601,2,1)……登録番号 8 の学校の 601 組の出席番号 2 番の生徒の住所

MUMPS のグローバル変数ではこのように蓄えようとするデータ項目の構造に応じて，添え字をいくつも付けることができるのです。

ローカル変数で試してみましょう。

```
>S HIZUKE(1789,7,14)="ﾌﾗﾝｽ ｶｸﾒｲ"↵

>S HIZUKE(1600,9,15)="ｾｷｶﾞﾊﾗ ﾉ ﾀﾀｶｲ"↵

>S HIZUKE(1954,7,28)="ﾎﾞｸ ﾉ ｾｲﾈﾝｶﾞｯﾋﾟ"↵

>S HIZUKE(1945,8,6)="ﾋﾛｼﾏ ｹﾞﾝﾊﾞｸ ﾄｳｶ"↵

>W HIZUKE(1600,9,15)↵
ｾｷｶﾞﾊﾗ ﾉ ﾀﾀｶｲ
>W HIZUKE(1945,8,6)↵
ﾋﾛｼﾏ ｹﾞﾝﾊﾞｸ ﾄｳｶ
>W HIZUKE(1789,7,14)↵
ﾌﾗﾝｽ ｶｸﾒｲ
>
```

変数名の前に『 ^ 』記号を付ければそのままグローバル変数でも同じことができます。

添え字をふたつ使用してデータをひとつのグローバル変数にまとめたルーチンを示します。

list 6-2

```
MB013   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1985.2.20;ﾃﾞｰﾀ ﾆｭｳﾘｮｸ (ｸﾞﾛｰﾊﾞﾙ ﾍﾝｽｳ ﾉ ﾘﾖｳ)
        ;
HAJIME  R !,"ﾊﾞﾝｺﾞｳ : ",NO
        R !,"ﾅﾏｴ ..... : ",NAMAE
        R !,"ｾｲﾍﾞﾂ .... : ",SEI
        R !,"ﾈﾝﾚｲ ..... : ",TOSHI
        R !,"ｼｮｿﾞｸ･ﾌﾞｼｮ : ",BUSHO
KAKUNOU S ^MBM(NO,0)=NAMAE
        S ^MBM(NO,1)=SEI
        S ^MBM(NO,2)=TOSHI
        S ^MBM(NO,3)=BUSHO
        Q
```

- グローバル変数とローカル変数には添え字を複数個付けることができる
- データは添え字を複数付けて，ひとつのグローバル変数にまとめて格納すると管理が容易である

4,データのリスト／検索ルーチンの訂正

前章で考えたデータのリスト，検索をするルーチン群をすべて改造してグローバル変数を用いるようにしましょう。

①指定された人のデータを表示するルーチン

list 6-3

```
MB014   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1985.2.19;ﾃﾞｰﾀ ﾉ ﾋｮｳｼﾞ
        ;
HAJIME  R !,"ﾊﾞﾝｺﾞｳ : ",NO
        W !,"ﾅﾏｴ ..... : ",^MBM(NO,0)
        W !,"ｾｲﾍﾞﾂ .... : ",^MBM(NO,1)
        W !,"ﾈﾝﾚｲ ..... : ",^MBM(NO,2)
        W !,"ｼｮｿﾞｸ･ﾌﾞｼｮ : ",^MBM(NO,3)
        Q
```

グローバル変数は WRITE 命令の中でも自由に使えます。

②すべての人のデータをリストするルーチン

list 6-4

```
MB015   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1985.2.19;ﾃﾞｰﾀ ﾉ ﾘｽﾄ･ｱｯﾌﾟ
        ;
HAJIME  R !,"ｻｲｺﾞ ﾉ ﾊﾞﾝｺﾞｳ ﾊ ? ",NN
        F J=1:1:NN W !,J,?5,^MBM(J,0),?20,^MBM(J,1),?25,^MBM(J,2),?30,^MBM(J,3)
        W !,"*** ｵﾜﾘ ***"
        Q
```

^MBM(NO,0) と ^MBM(J,0) は NO=J ならば同じ変数であることに注意してください。NO もＪも添え字の値を表す変数ですから，変数名よりその値が問題になります。必要なら ^MBM(X,0) や ^MBM(ABC,0) などと表してもかまいません。

X=3 のとき，^MBM(X,0) は登録番号３番の人の名前です。
ABC=55 のとき ^MBM(ABC,1) は登録番号55の人の性別です。

③女性のみのリスト

list 6-5

```
MB016   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1985.2.19;ｼﾞｮｾｲ ﾉﾐ ﾉ ﾃﾞｰﾀ ﾉ ﾘｽﾄ･ｱｯﾌﾟ
        ;
HAJIME  R !,"ｻｲｺﾞ ﾉ ﾊﾟｿｺﾞｳ ﾊ ? ",NN
        F J=1:1:NN I ^MBM(J,1)="F" W !,J,?5,^MBM(J,0),?20,^MBM(J,1),?25,^MBM(J,2
    -->),?30,^MBM(J,3)
        W !,"*** ｵﾜﾘ ***"
        Q
```

^MBM(J,1) には J 番目の人の性別が入っています。したがって

```
I ^MBM(J,1)="F" W !,J,...
```

で J 番目の人の性別が 'F' すなわち女性の場合にのみ，右側の WRITE 命令を実行します。

④男性のみのリスト

list 6-6

```
MB017   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1985.2.19;ﾀﾞﾝｾｲ ﾉﾐ ﾉ ﾃﾞｰﾀ ﾉ ﾘｽﾄ･ｱｯﾌﾟ
        ;
HAJIME  R !,"ｻｲｺﾞ ﾉ ﾊﾟｿｺﾞｳ ﾊ ? ",NN
        F J=1:1:NN I ^MBM(J,1)="M" W !,J,?5,^MBM(J,0),?20,^MBM(J,1),?25,^MBM(J,2
    -->),?30,^MBM(J,3)
        W !,"*** ｵﾜﾘ ***"
        Q
```

⑤30才未満の人のデータのリスト

list 6-7

```
MB018   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1985.2.19;30 ｻｲ ﾐﾏﾝ ﾉ ﾋﾄ ﾉ ﾃﾞｰﾀ ﾉ ﾘｽﾄ･ｱｯﾌﾟ
        ;
HAJIME  R !,"ｻｲｺﾞ ﾉ ﾊﾟｿｺﾞｳ ﾊ ? ",NN
        F J=1:1:NN I ^MBM(J,2)<30 W !,J,?5,^MBM(J,0),?20,^MBM(J,1),?25,^MBM(J,2)
    -->,?30,^MBM(J,3)
        W !,"*** ｵﾜﾘ ***"
        Q
```

^MBM(J,2) には J 番目の人の年齢が入っています。

```
I ^MBM(J,2)<30
```

は「もしJ番目の人の年齢が30より小さければ」という条件になります。

⑥指定された年齢未満のデータのリスト

list 6-8

```
MB019    ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1985.2.19;ﾆﾝｲ ﾉ ﾈﾝﾚｲ ﾐﾏﾝ ﾉ ﾋﾄ ﾉ ﾃﾞｰﾀ ﾉ ﾘｽﾄ･ｱｯﾌﾟ
         ;
HAJIME   R !,"ｻｲｺﾞ ﾉ ﾊﾝｺﾞｳ ﾊ ? ",NN
         R !,"ﾅﾝｻｲ ﾐﾏﾝ ﾉ ﾋﾄ ﾉ ﾃﾞｰﾀ ｦ ﾘｽﾄ ｼﾏｽｶ ? ",TOSHI
         F J=1:1:NN I ^MBM(J,2)<TOSHI W !,J,?5,^MBM(J,0),?20,^MBM(J,1),?25,^MBM(J
      -->,2),?30,^MBM(J,3)
         W !,"*** ｵﾜﾘ ***"
         Q
```

⑦名前による検索

list 6-9

```
MB020    ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1985.2.19;ﾅﾏｴ ﾆ ﾖﾙ ｹﾝｻｸ
         ;
HAJIME   R !,"ｻｲｺﾞ ﾉ ﾊﾝｺﾞｳ ﾊ ? ",NN
         R !,"ﾅﾏｴ ﾊ ? ",NAMAE
         F J=1:1:NN I ^MBM(J,0)=NAMAE W !,J,?5,^MBM(J,0),?20,^MBM(J,1),?25,^MBM(J
      -->,2),?30,^MBM(J,3)
         W !,"*** ｵﾜﾘ ***"
         Q
```

⑧名前の一部による検索

list 6-10

```
MB021    ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1985.2.19;ﾅﾏｴ ﾆ ﾖﾙ ｹﾝｻｸ
         ;
HAJIME   R !,"ｻｲｺﾞ ﾉ ﾊﾝｺﾞｳ ﾊ ? ",NN
         R !,"ﾅﾏｴ ﾊ ? ",NAMAE
         F J=1:1:NN I ^MBM(J,0)[NAMAE W !,J,?5,^MBM(J,0),?20,^MBM(J,1),?25,^MBM(J
      -->,2),?30,^MBM(J,3)
         W !,"*** ｵﾜﾘ ***"
         Q
```

- 変数の添え字にはどのような変数を用いてもよい。すなわち X＝1, Y＝3, ABC＝0, NO＝1のとき, ^MBM(X,0), ^MBM(NO,0), ^MBM(Y-2,0), ^MBM(ABC＋1,ABC)はすべて同じ変数である

5,あらゆるグローバル変数に使える グローバルリストユーティリティ

これまで入力したデータはすべて専用のルーチンを作成してリストアップしてきました。

登録されているデータを表示/印刷するということはあらゆる業務においてもっとも基本的な作業です。

しかし，あるグローバル変数専用のリストアップルーチンを作っておれないことがあります。またルーチンの誤りのためデータが思ったところへ登録されず，表示しようと思ってもできなかったりすることがあります。

MUMPS にはディスクに登録されているデータを簡単に調べるためのルーチン %GL（グローバルリストユーティリティ）が用意されています。

```
>D ^%GL↵
```

としてください。すると

```
グローバル　リスト　ユーティリティ

グローバルの選択: ^
```

と表示されるので，ここで内容を調べたいグローバル変数名を入力します。

```
グローバルの選択: ^MBM↵
```

すると，次に

```
出力装置 <0>
スクロールさせますか? <N>
```

というふたつの質問が現れます。

今はここではどちらにも，↵キーのみを打ってください。すると，指定したグローバル変数のリストが表示されます。

```
^MBM("1","0")="イワサキ ヒロミ"
^MBM("1","1")="F"
^MBM("1","2")="26"
^MBM("1","3")="コウホウシツ"
^MBM("2","0")="ヤマダ タロウ"
^MBM("2","1")="M"
```

：%GL でグローバル変数をリストすると，変数の添え字ひとつ1つに引用符「"」が付きますが，気にしないでください。^MBM("1","0")は^MBM(1,0)のことです

```
^MBM("2","2")="19"
^MBM("2","3")="ﾍﾝｼｭｳﾌﾞ"
^MBM("3","0")="ｼﾏ ﾖｼﾅﾘ"
   :
   :
```

そしてこのリストが終わると再び

グローバルの選択：^

という質問が現われるので，必要ならまたグローバル変数名を指定してください。もし↵キーのみを打つとこれでこのルーチンは終了します。

『？』記号をタイプするとこのルーチンの使用法の案内が表示されるので，参考にしてください。

- グローバル変数の内容を調べるにはルーチン %GL を用いると便利である

6,ルーチンを呼び出すときの注意

グローバルリストユーティリティを呼び出して実行するには

```
>D ^%GL↵
```

としましたが，これは

```
>ZL %GL D %GL↵
```

と同じことです。ディスク上に登録されているルーチンを呼び出して実行するときは，DO 命令で『^』記号を付けてそのルーチン名を指定するだけでよいのです。ちょうどディスク上の変数（すなわちグローバル変数）には名前の前に『^』記号を付けるのとよく似ています。今までディスクに登録してあるルーチンやグローバル変数の名前を調べるのに

```
>D ^%RD↵        ：ルーチン名を調べる
```

とか

```
>D ^%GD↵        ：グローバル名を調べる
```

としてきましたが，これらはディスク上の %RD と %GD というルーチンを呼び出して実行していたのです。

ただし DO 命令でルーチンを呼び出すときには，ZLOAD 命令のときと同

じような注意が必要です。

ワークエリア上に他のルーチンが存在していると，DO 命令も ZLOAD 命令も，ワークエリア上のルーチンを消去してしまいます。もしディスクに登録していないルーチンがワークエリア上にあればそれを登録してからこれらの命令を用いてください。

- ディスク上のルーチン ABC を呼び出して実行するには，D ^ABCとすることができる。これは'ZL ABC D ABC' と同じことである
- D ^ルーチン名で，ディスク上のルーチンを呼び出すとき，ワークエリア上にあるルーチンが，ワークエリア上から消去される

7, 合計と平均

データリストのルーチンの応用で，登録されている人の平均年齢を計算しましょう。

平均は，合計から算出するので，まず合計を求めることから解説します。

MUMPS で合計を求める方法はひとつの「定石」(決まりきった手順) なので覚えてください。

list 6-11

```
MB022   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1986.2.12;ﾈﾝﾚｲ ﾉ ｺﾞｳｹｲ
        ;
HAJIME  R !,"ｻｲｺﾞ ﾉ ﾊﾞﾝｺﾞｳ ﾊ ? ",NN
        S GOKEI=0
        F J=1:1:NN S GOKEI=GOKEI+^MBM(J,2)
        W !,"ﾈﾝﾚｲ ｺﾞｳｹｲ = ",GOKEI
        Q
```

まず，実行します。

```
>D MB022↵

ｻｲｺﾞ ﾉ ﾊﾞﾝｺﾞｳ ﾊ ? 5↵
ﾈﾝﾚｲ ﾉ ｺﾞｳｹｲ = 138          ：登録されている人全員の年齢の合計です
>
```

さて，このルーチンの主たる手順は

①最初に GOKEI を0にセットすること

②S GOKEI=GOKEI+^MBM(J,2) という命令

です。

②から解説します。

```
S GOKEI=GOKEI+^MBM(J,2)
```

は，GOKEI+^MBM(J,2) が GOKEI と等しいといっているのではありません。SET 命令は，まず「=」の右側の式(GOKEI+^MBM(J,2)) を計算します。

たとえば

```
GOKEI=53    J=5    ^MBM(5,2)=32
```

とすると

```
GOKEI+^MBM(J,2)=GOKEI+^MBM(5,2)
               =53+32
               =85
```

になります。これを「=」の左にある変数にセットするので

```
S GOKEI=85
```

という意味になります。

結局，変数 GOKEI の値が53から85に増加した（GOKEI に，^MBM(5,2) の値を加えた）ことになります。

この手順をすべてのJについて実行すれば，変数 GOKEI には^MBM(J,2) の合計がセットされることになります。

①の S GOKEI=0 ですが，これは簡単です。

ソロバンで合計を計算する場合でも，最初に「ご破算」にしてソロバンの玉を0の状態にします。MUMPS で合計する場合でも，まず'合計'を入れる箱を空にしておくのです。

さもないと前に計算した GOKEI の値にさらに加算した結果が出るか，あるいは，変数 GOKEI が定義されていないため

```
S GOKEI=GOKEI+^MBM(J,2)
```

この GOKEI 部分で 〈UNDEF〉 エラーになってしまいます。

さて合計が出たので，平均を算出するルーチンを掲げておきます。平均は合計を人数で割っただけです。

list 6-12

```
MB023   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1986.2.12;ﾍｲｷﾝ ﾈﾝﾚｲ ﾉ ｹｲｻﾝ
        ;
HAJIME  R !,"ｻｲｺﾞ ﾉ ﾊﾞﾝｺﾞｳ ﾊ ? ",NN
        S GOKEI=0
        F J=1:1:NN S GOKEI=GOKEI+^MBM(J,2)
        W !,"ﾍｲｷﾝ ﾈﾝﾚｲ = ",GOKEI/NN
        Q
```

ルーチンを入力して，実行してください。

```
>D MB023↵

ｻｲｺﾞ ﾉ ﾊﾞﾝｺﾞｳ ﾊ ? 5↵
ﾍｲｷﾝ ﾈﾝﾚｲ = 27.6          ：平均年齢が表示されます
>
```

本論からはそれますが，キーボードから入力したデータの合計と平均を算出するルーチンの例を掲げておきましょう。

list 6-13

```
YSTST2  ;YS,SAMPLE;1986.2.12;ｺﾞｳｹｲ ﾄ ﾍｲｷﾝ ﾉ ｹｲｻﾝ
        ;
KAZU    R !,"# ﾃﾞｰﾀ ﾉ ｺｽｳ ? ",KAZU
DATA    F J=1:1:KAZU W !,"ﾃﾞｰﾀ (",J,") ? " R DATA(J)
KEISAN  S TOTAL=0
        F J=1:1:KAZU S TOTAL=TOTAL+DATA(J)
HYOJI   W !,"ｺﾞｳｹｲ = ",TOTAL,"   ﾍｲｷﾝ = ",TOTAL/KAZU
        Q
```

- S X＝X＋A は，X＋A と X が等しいという意味ではなく，X に A を加えるという意味である
- データの合計を算出するルーチンの「定石」：

  S TOTAL＝0

  F J＝1,1:KAZU S TOTAL＝TOTAL＋DATA (J)

8,繰り返し——FOR命令とDO命令を組み合わせる

もう少し仕事を進めるためにルーチン MB013 を用いて，さらに何人かデータを登録してください。

```
>D ^MB013

ﾊﾞﾝｺﾞｳ : 6
ﾅﾏｴ ..... : ﾊｼﾓﾄ ｱﾂｺ
ｾｲﾍﾞﾂ .... : F
ﾈﾝﾚｲ ..... : 22
ｼｮｿﾞｸ･ﾌﾞｼｮ : ﾀﾞｲ2 ｴｲｷﾞｮｳｶﾌﾞ
>D ^MB013

ﾊﾞﾝｺﾞｳ : 7
ﾅﾏｴ ..... : ﾋﾗﾀ ﾐﾈｺ
ｾｲﾍﾞﾂ .... : F
ﾈﾝﾚｲ ..... : 22
ｼｮｿﾞｸ･ﾌﾞｼｮ : ｿｳﾑｶ
>D ^MB013
   :
   .
```

こうして何人ものデータを入力していると，いちいち 'D ^MB013' とタイプすることが苦痛になってきます。そこで，繰り返してデータを登録することのできるルーチンを考えてみましょう。

list 6-14

```
MB024   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1985.3.14;ﾃﾞｰﾀ ﾆｭｳﾘｮｸ (ｸﾞﾛｰﾊﾞﾙ ﾍﾝｽｳ ﾉ ﾘﾖｳ)
        ;
KAISHI  R !,"ﾅﾝﾆﾝ ﾄｳﾛｸ ｼﾏｽｶ ? ",N
        F K=1:1:N D TOROKU ←――― ①この行に注目
        Q ←――――――――――― ②この行に注目
TOROKU  R !,"ﾊﾞﾝｺﾞｳ : ",NO
        R !,"ﾅﾏｴ ..... : ",NAMAE
        R !,"ｾｲﾍﾞﾂ .... : ",SEI
        R !,"ﾈﾝﾚｲ ..... : ",TOSHI
        R !,"ｼｮｿﾞｸ･ﾌﾞｼｮ : ",BUSHO
KAKUNOU S ^MBM(NO,0)=NAMAE
        S ^MBM(NO,1)=SEI
        S ^MBM(NO,2)=TOSHI
        S ^MBM(NO,3)=BUSHO
        Q
```

このルーチンは，行 KAISHI+1 ① の FOR 命令と DO 命令の働きで，行 TOROKU 以降を繰り返して実行します。

FOR 命令は，その右側にある命令を繰り返す命令です。今までの例では，FOR 命令の右側には WRITE 命令がありました。

```
F J=1:1:N W !,J,?5,^MBM(J,0),?20,^MBM(J,1),?25,^MBM(J,2),?30,^MBM(J,3)
          WRITE命令です
```

このルーチンでは FOR 命令の右には D TOROKU という DO 命令があります。D TOROKU という命令は行 TOROKU からそのルーチンの最後までを実行することを指示しています。一方 F K=1:1:N は,「変数Kの値が最初は1，それを1ずつ増やしてNになるまで（N回）繰り返す」ということです。したがって

```
F K=1:1:N D TOROKU
```
FOR命令　　DO命令

は，行 TOROKU 以降をN回繰り返すことになります。

行 TOROKU 以降を1回実行すると1人のデータを登録することができますから，このルーチン全体では，N人のデータを入力することができることになります。

ところで行 KAISHI+2（②）の QUIT 命令は何のために入っているのでしょうか。

QUIT 命令は，ルーチンの実行をそこで終わらせます。もしこの命令がないとどうなるでしょう？

ここの QUIT 命令を取り除いてみました

```
KAISHI   R !,"ﾅﾝﾆﾝ ﾄｳﾛｸ ｼﾏｽｶ ? ",N
         F K=1:1:N D TOROKU  <----------- ③
TOROKU   R !,"ﾊﾞﾝｺﾞｳ : ",NO  <----------- ④
         R !,"ﾅﾏｴ ..... : ",NAMAE
         R !,"ｾｲﾍﾞﾂ .... : ",SEI
         R !,"ﾈﾝﾚｲ ..... : ",TOSHI
         R !,"ｼｮｿﾞｸ･ﾌﾞｼｮ : ",BUSHO
KAKUNOU  S ^MBM(NO,0)=NAMAE
         S ^MBM(NO,1)=SEI
         S ^MBM(NO,2)=TOSHI
         S ^MBM(NO,3)=BUSHO
         Q  <----------- ⑤
```

行 KAISHI+1 ③ で D TOROKU をN回繰り返すとこの行が終わり，次の行へ移ります。するとそこは行 TORUKU ④ ですから，またデータ入力が始まります。

1人分のデータを入力し，それをグローバル変数にセットしたあと，ルーチンの最後の QUIT 命令⑤に達して，ようやくここでルーチンの実行が終わります。結局のところ実際は，行 TOROKU 以降のデータ入力部分を N+1 回繰り返したことになります。

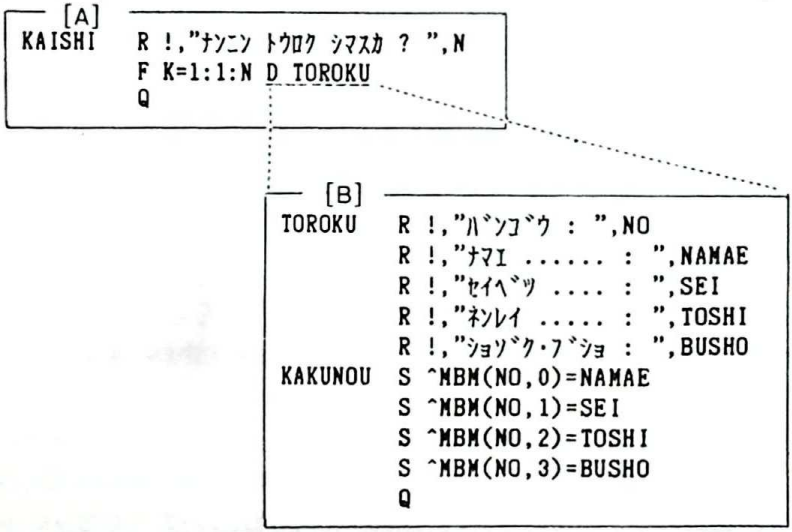
一方，行 KAISHI+1 ③ の次の行に QUIT 命令があると，そこでルーチンの実行が終了します。すると行 TOROKU 以降は，D TOROKU で呼び出されたときにしか実行されません。

このようなルーチンの途中にある QUIT 命令を，初心者はよく忘れるようです。これがないと，余分な作業をすることになります。注意しましょう。

- FOR 命令と DO 命令の組み合わせて，ルーチンの一部を繰り返して実行することができる
- QUIT 命令がルーチンの途中にあると，そのルーチンはその QUIT 命令のところで終了する
- QUIT 命令がないと，ルーチンを余分に実行してしまうことがある

9,仕事のレベル差のコントロール　QUIT命令

もう少しよく考えてみましょう。行 KAISHI から行 KAISHI+2 を［ A ］部分，行 TOROKU から最後までを［ B ］部分として図解してみます。



［ B ］部分は［ A ］部分の中の D TOROKU を拡大したものといえます。

［Ｂ］はひとり１人のデータを入力する部分，［Ａ］はそれを何回繰り返すかを制御する部分です。したがって［Ａ］と［Ｂ］は「仕事のレベル」が異なると考えられます。

このようにひとつのルーチンの中には，しばしば，仕事のレベルが異なる部分があります。DO 命令は違うレベルの仕事を呼び出すときに用いられます。そのようなときに，レベルの異なる部分の間には必ず QUIT 命令を入れると覚えておいてください。

- ひとつのルーチンの中には，レベルの違う仕事がある
- DO 命令は，レベルの違う仕事を呼び出す
- ひとつのルーチンで，レベルの違う仕事をする部分があるときには，それらの間に QUIT 命令を入れておく

10,ルーチンを強制中断する

前出のルーチン MB024 では，MUMPS は最初に何人分のデータを登録するかと質問してきます。ここで，予定の人数より大きな数を入れてしまったらどうなりますか？　たとえば

```
ﾅﾝﾆﾝ ﾄｳﾛｸ ｼﾏｽｶ ? 99999↵
```

と入力した場合，99999人分データを登録しないとこのルーチンは終わりません。多分こんなにたくさんデータはないでしょうから，これでは困ったことになります。

そのようなとき，キーボードの左隅にある CTRL というキーを押さえながら，アルファベットのＣキーを押さえてください。すると

```
<INRPT> 割込み要求が発生しました
```

というエラーメッセージが出てルーチンの実行が中断されます。

この方法は CTRL/C（コントロール・シー）と呼び，MUMPS の仕事を強制的に中断したいときにいつでも使えます。

困ったときの「CTRL/C」頼み！

- CTRLキーを押さえながらCキーを押さえる（CTRL/C）とMUMPSの仕事を強制中断できる

11，あらかじめ繰り返しの回数を指定しない方法

予定の人数より大きな数を指定しておいて，強制中断を使えばあらかじめ何人の登録をするかを指定せずに済みます。

それには FOR 命令の回数を，わざわざ非常に大きくしておきます。

リスト6-15のルーチンは行 TOROKU 以降を999999回繰り返します。

しかし

```
ﾊﾞﾝｺﾞｳ :
```

という質問が出たときに，CTRL/C で強制中断すれば，そこで仕事を止めることができます。

あらかじめ何人登録するかを考えずに済むので便利です。

ただ，これは余りスマートな方法とはいえません。

QUIT 命令をじょうずに利用すると，もっとスマートなルーチンにすることができます。

リスト6-16がそれです。

list 6-15

```
MB025   ;YS,ｼｮｸｲﾝﾒｲﾎﾞ;1985.3.14;ﾃﾞｰﾀ ﾆｭｳﾘｮｸ
        ;
KAISHI  F K=1:1:999999 D TOROKU
        Q                          ←ここに注目
TOROKU  R !,"ﾊﾞﾝｺﾞｳ : ",NO
        R !,"ﾅﾏｴ ..... : ",NAMAE
        R !,"ｾｲﾍﾞﾂ .... : ",SEI
        R !,"ﾈﾝﾚｲ ..... : ",TOSHI
        R !,"ｼｮｿﾞｸ･ﾌﾞｼｮ : ",BUSHO
KAKUNOU S ^MBM(NO,0)=NAMAE
        S ^MBM(NO,1)=SEI
        S ^MBM(NO,2)=TOSHI
        S ^MBM(NO,3)=BUSHO
        Q
```

list 6-16

```
MB026   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1985.3.14;ﾃﾞｰﾀ ﾆｭｳﾘｮｸ
        ;
KAISHI  F K=1:1:999999 D TOROKU I NO="*" Q
        Q
TOROKU  R !,"ﾊﾞﾝｺﾞｳ : ",NO
     ② I NO="*" Q
        R !,"ﾅﾏｴ ..... : ",NAMAE
        R !,"ｾｲﾍﾞﾂ .... : ",SEI
        R !,"ﾈﾝﾚｲ ..... : ",TOSHI
        R !,"ｼｮｿﾞｸ･ﾌﾞｼｮ : ",BUSHO
KAKUNOU S ^MBM(NO,0)=NAMAE
        S ^MBM(NO,1)=SEI
        S ^MBM(NO,2)=TOSHI
        S ^MBM(NO,3)=BUSHO
        Q
```

①

ここに注目！

①と②のIF命令とQUIT命令に注目してください

行 KAISHI の D TOROKU で，行 TOROKU 以降が呼び出されます。'バンゴウ：' という質問が表示され，キーボードから入力したデータが NO という変数にセットされます。

1)ここで何か番号を入力すると，②の部分

```
I NO="*" Q
②
```

この QUIT 命令は無視される

の条件 NO="*" は正しくない（「偽」である）ので，その右の QUIT 命令は実行されません。そして以降の行の READ 命令，SET 命令へと続きます。

ルーチン最後の QUIT 命令に達すると，行 KAISHI の DO 命令 D TOROKU が終了したことになります。

```
KAISHI    F K=1:1:999999 D TOROKU I NO="*" Q
          Q
```

呼び出す

ここへ戻る

```
TOROKU    R !,"ﾊﾞﾝｺﾞｳ : ",NO
          I NO="*" Q
          R !,"ﾅﾏｴ ..... : ",NAMAE
             :
             :
KAKUNOU   S ^MBM(NO,0)=NAMAE
             :
             :
          Q
```

ここでD TOROKUが終了

すると MUMPS は DO 命令の右にある IF 命令①

```
KAISHI  F K=1:1:999999 D TOROKU I NO="*" Q
                                  ①
```

この QUIT 命令も
無視されます

を実行します。しかしここでも NO には番号が入っているのでこの条件は「偽」ですからこの行の最後の QUIT は無視されます。

これで FOR 命令による繰り返しが1回終了したことになります。

FOR 命令の回数は十分大きな数なので，また繰り返しが始まります。そして D TOROKU が実行され，行 TOROKU 以降が呼び出されます。

```
KAISHI   F K=1:1:999999 D TOROKU I NO="*" Q
         Q
                  呼び出す
TOROKU   R !,"ﾊﾞﾝｺﾞｳ : ",NO
           :
```

2)ところが，行 TOROKU が呼び出されたとき，番号の代わりに，『＊』記号を入力すると今述べたこととは違ったことが起こります。

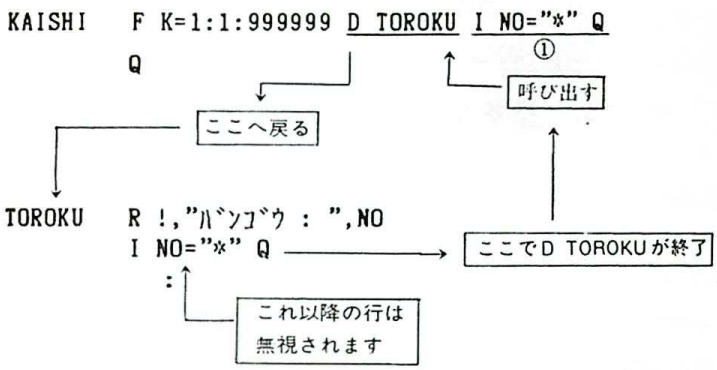
入力した『＊』記号は変数 NO にセットされます。

すると次の行の

```
I NO="*" Q
  ②
```

②の IF 命令の条件が正しく（「真」に）なり，IF 命令の右にある QUIT 命令が実行されます。

QUIT 命令は「ルーチンはここで終わり」という意味で，これが実行されると，それ以降の行は無視されます。もともと行 TOROKU 以降は，D TOROKU という DO 命令で呼び出されていたので，その呼び出しが終了したことになります。



DO 命令の呼び出しが終了すると，MUMPS はその DO 命令の次の命令を実行しはじめます。ここでは DO 命令の右側にある①の IF 命令を実行します。

```
KAISHI    F K=1:1:999999 D TOROKU I NO="*" Q
          Q                              ①
```

その IF 命令もまた，NO＝"*" という条件になっています。さきほど変数 NO には『＊』を入力したので，やはりこの条件も「真」です。このためさらに，その右にある QUIT 命令が実行されます。

ところで，この QUIT 命令は FOR 命令を終了させる働きがあるのです。そのため FOR 命令による繰り返しが終わります。FOR 命令が終わればこの行全体が終了したことになり，次の行へ移ります。次の行 KAISHI+1 にはもうひとつ QUIT 命令があります。

この QUIT 命令には，終了させるべき DO 命令も FOR 命令もないので，このルーチン全体を終了させます。

これで強制中断しなくても，好きなときに登録を終了するルーチンを作るこ

とができました。

QUIT 命令にはここで解説したように，いろいろな働きがあります。

ⓐ QUIT 命令の左に FOR 命令があるときには，その FOR 命令を終わらせる

ⓑ FOR 命令がないときには，その行を呼び出した DO 命令を終わらせる

ⓒ 終わらせるべき DO 命令もないときにはルーチン全体を終わらせる

ルーチンを実行するときには

```
>D MB026↵
```

というように，DO 命令を用いているのですから，ⓑとⓒは同じものと考えることもできます。

また DO 命令で

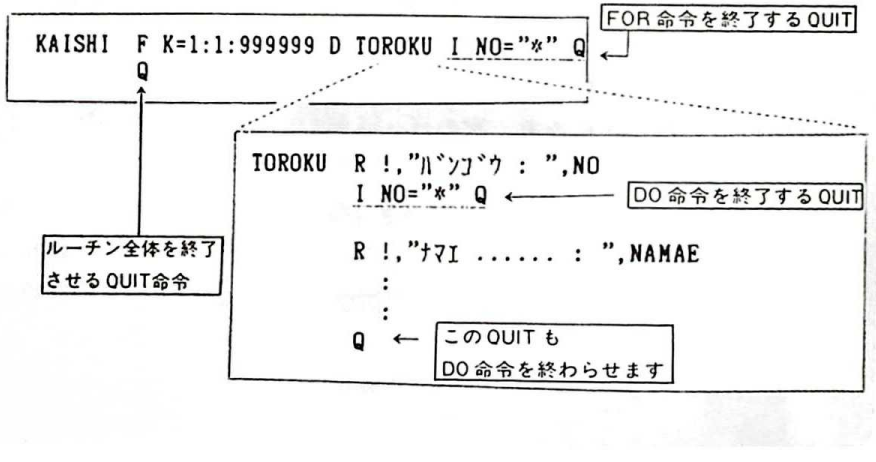
```
>D MB026↵
```

とルーチンを呼び出したとき，そのルーチンが終了するとまた

```
>
```

という MUMPS に命令を与える状態に戻ります。

これと同じように，ルーチンの中に DO 命令があるときは，呼び出された部分が終了すると，その DO 命令の次の命令に戻るということも覚えてください。



- QUIT 命令は，その左側に FOR 命令があるとその FOR 命令を終了させる
- FOR 命令と関係のない QUIT 命令は，ルーチンのその部分を呼び出した DO 命令を終了させる
- FOR 命令とも DO 命令とも関係のない QUIT 命令はルーチン全体を終了させる
- ある処理を任意回数繰り返すルーチンの構造：

```
        F K=1:1:99999 D 処理 I 終了条件 Q
        Q
処理    ....
        I 終了条件 Q
        ....
```

12，その他のルーチンの例

①数値の 2 乗を計算するルーチン

list 6-17

```
YSTST3  ;YS;1986.2.2; ﾙｰﾌﾟ ﾉ ﾚｲ
        ;
HAJIME  F K=1:1:10000000 D KEISAN I DATA="*" Q
        Q
KEISAN  R !,"DATA ? ",DATA
        I DATA="*" Q
        W !,DATA,"*",DATA,"=",DATA*DATA
        Q
```

FOR 命令の 10000000 は十分大きい数であれば，何でも構いません。

②データが10より大きいかどうかを判定するルーチン

リスト6-18のように 'X=' という質問に END と入力します。

するとこのルーチンは終了します。また，終了前に 'オワリ' と表示することにも注意してください。

list 6-18

```
YSTST4   ;YS;1986.2.2;ﾙｰﾌﾟ ﾉ ﾚｲ
         ;
LOOP     F NN=1:1:9999999 D HANTEI I X="END" Q
         Q
HANTEI   R !,"X = ",X
         I X="END" W !,"ｵﾜﾘ" Q
         I X>10 W !,X," ﾊ 10 ﾖﾘ ｵｵｷｲ"
         I X<10 W !,X," ﾊ 10 ﾖﾘ ﾁｲｻｲ"
         I X=10 W !,X," ﾊ 10 ﾆ ﾋﾄｼｲ"
         Q
```

前節のルーチン MB026（リスト6-16）も，この節のふたつのルーチンも，処理を終了させるために同じ終了条件と QUIT 命令を２か所ずつ書いてあることに注目してください。

ひとつは FOR 命令による繰り返しを終了させるのに必要で，もうひとつは DO 命令による呼び出しを終了させるのに必要です。

Step 3

オペレーティングMUMPS

コンピュータに登録するデータには，しばしば登録番号をつけます。登録番号が連続していると都合が良いのですが，現実には普通とびとびの値をとります。

Chapter7では，データの削除の方法の関連と，MUMPSデータベースにおけるデータの親子関係という概念を説明します。

Chapter8ではとびとびの登録番号をもつデータベースの操作方法とMUMPSデータベースの兄弟関係について述べます。

Chapter9では前章の応用としてデータを並べ替えてリストする方法を解説します。

Chapter 10では職員名簿管理をより便利なものにする方法を紹介します。そのなかでプログラムのモジュール化という考え方を解説しましょう。

Chapter 7

データを削除する

データを登録するルーチンやリストするルーチンを作ってきましたが，不必要なデータの「削除」という機能も実際の業務上必要なものです。この章では，データの削除の方法とデータがあるかどうかを調べる方法，それに伴ういくつかの問題点について解説します。

1,データの削除

登録されているデータを削除するには KILL 命令を使います。

たとえば，ローカル変数で試してみましょう。

```
>S XX=123↵      ：変数XXをセットしました

>W XX↵          ：内容を確かめてみます
123
>K XX↵          ：変数XXを削除しました

>W XX↵          ：内容を確かめます

<UNDEF> 定義されていない変数が参照されました
 0)*XECUTE*:: W XX                          ：エラーです
<UNDEF>*XECUTE*:::4:0 W XX
>
```

KILL された変数を表示しようとすると 〈UNDEF〉 エラーになってしまいます。

KILL 命令はグローバル変数や，添え字付き変数にも用いることができます。

```
>S ^GLOBAL(1,1)="ABC"↵

>S ^GLOBAL(1,2)="XXX"↵
```

```
>S ^GLOBAL(2,3,4)="MUMPS"↵

>S ^GLOBAL(2,3,5)="TEST"↵

>S ^GLOBAL(2,10)="ZZZZ"↵

>S ^GLOBAL(5)="DATA"↵
```

とセットしておいてから，ルーチン %GL を用いて内容を調べます。

```
>D ^%GL↵

グローバル　リスト　ユーティリティ

                                    ：表示する変数名を指定します
グローバルの選択：^GLOBAL↵
出力装置 <0> ↵                      ：↵のみを打ってください
スクロールさせますか? <0> ↵         ：↵のみを打ってください
^GLOBAL("1","1")="ABC"
^GLOBAL("1","2")="XXX"
^GLOBAL("2","3","4")="MUMPS"
^GLOBAL("2","3","5")="TEST"
^GLOBAL("2","10")="ZZZZ"
^GLOBAL("5")="DATA"

グローバルの選択：^↵               ：ここで↵キーのみを打つと，
>                                   このプログラムは終了します
```

次に適当に変数を消してしまいます。

コンマに注意

```
>K ^GLOBAL(2,3,5),^GLOBAL(1,1)↵
```

そして %GL でもう一度変数を調べてください。

```
^GLOBAL("1","2")="XXX"
^GLOBAL("2","3","4")="MUMPS"
^GLOBAL("2","10")="ZZZZ"
^GLOBAL("5")="DATA"
```

と表示されましたか？　^GLOBAL(2,3,5) と ^GLOBAL(1,1) が削除されたため表示されていないことに注意してください。

同様に，職員名簿から登録番号 NO 番の人のデータを削除するには

```
K ^MBM(NO,0),^MBM(NO,1),^MBM(NO,2),^MBM(NO,3)
```

とすればよいことは容易に理解できると思います。

- KILL 命令は変数を削除する
- 削除された変数は存在しないので，その値を表示しようとすると，〈UNDEF〉というエラーになる

2，変数の親子関係

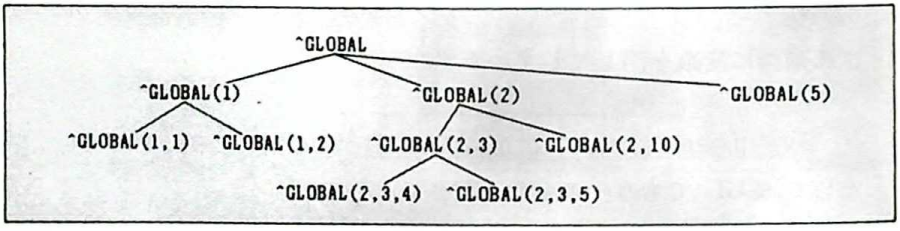
MUMPS の添え字付き変数は，グローバル変数もローカル変数も「親子関係」で管理されています。

前節の ^GLOBAL という例を考えます。いま

```
^GLOBAL(1,1)="ABC"
^GLOBAL(1,2)="XXX"
^GLOBAL(2,3,4)="MUMPS"
^GLOBAL(2,3,5)="TEST"
^GLOBAL(2,10)="ZZZZ"
^GLOBAL(5)="DATA"
```

というグローバル変数がセットされました。すると MUMPS は，これらの変数を次の fig 7-1 のように関連付けて記憶します。

fig 7-1　変数の構成



この構成を親子関係として考えると理解しやすいと思います。

^GLOBAL(2,3,4)　は ^GLOBAL(2,3)　　の「子供」である
^GLOBAL(2,3)　　は ^GLOBAL(2)　　の「子供」である
^GLOBAL(2)　　は ^GLOBAL　　の「子供」である
^GLOBAL(1)　　は ^GLOBAL(1,2)，^GLOBAL(1,3) などの「親」である
^GLOBAL　　は ^GLOBAL という名前のすべての変数の「親」である

という関係です。図の上の方を「親」と，下の方を「子供」と考えます。一般には「親」のことを「上位節」，「子供」のことを「下位節」と呼びます。

ところで，この図には ^GLOBAL，^GLOBAL(1)，^GLOBAL(2)，^GLOBAL(2,3) という変数が記憶されていますが，この 4 つの変数はセットした覚えがありません。

実はこの 4 つの変数は，MUMPS が暗黙のうちに作った変数です。^GLOBAL(1,2)，^GLOBAL(2,3,4) などの「子供」がセットされたとき，その「親」がいないと親子関係で管理することができません。そのため，MUMPS は値のない「仮想的な親」を自動的に作ります。

「仮想的な親」変数には値がないので，これを表示しようとするとエラーになります。

```
>W ^GLOBAL(2)↵

<UNDEF> 定義されていない変数が参照されました
 0) *XECUTE*:: W ^GLOBAL
<UNDEF>*XECUTE*:::4:1 W ^GLOBAL
```

今までの知識では，この「仮想的な親」の存在を知る方法はありません（その方法については後述します）。またなぜこのようなものが必要かは納得しにくいかもしれません。しかし，これは MUMPS でデータ処理をするために重要な概念なので，このまま覚えてください。

- 添え字付き変数は，親子関係で管理されている
- 左端から同じ添え字を持っている変数で，添え字の数の少ない変数を「上位節」または「親」と呼び，数の多い変数を「下位節」または「子供」という
- 「親」がないところへ，「子供」だけをセットすると，その「子供」を管理するために，「仮想的な親」が自動的にセットされる
- 「仮想的な親」変数はそれ自身には値がないので，値を参照しようとすると〈UNDEF〉エラーになる

3，親亀こけると子亀もこける

前節のグローバル変数 ^GLOBAL がセットされている状態で

```
>K ^GLOBAL(2)↵
```

としてください。

そしてルーチン %GL を用いてグローバル変数の内容を調べましょう。すると，^GLOBAL(2) の「子供」にあたる変数 ^GLOBAL(2,3,4) や ^GLOBAL(2,10) などが削除されているはずです。

「親」変数が削除されると，それが「仮想的な親」であっても，その「子供」もすべて削除されてしまうのです。この

親亀がこけると子亀もこける

という簡単な規則を理解してください。「子供」が削除されれば，「孫」も削除されることは当然です。

また，この規則はグローバル変数でもローカル変数でもまったく同じことです。

このことを利用すると「職員名簿」のデータの削除が簡単になります。たとえば

```
^MBM(12)
```

という変数を削除すると

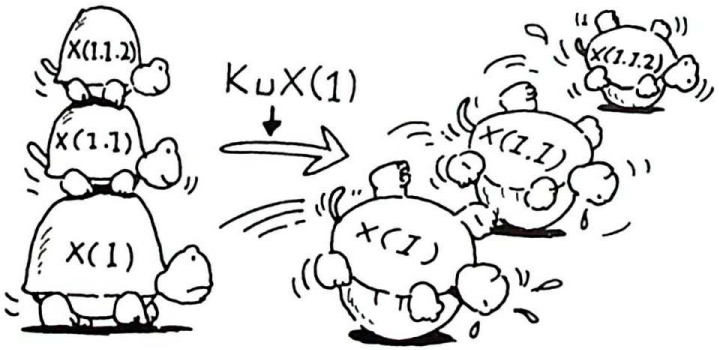
```
^MBM(12,0)   ^MBM(12,1)   ^MBM(12,2)   ^MBM(12,3)
```

という変数がすべて削除されてしまうのです。

したがって，職員名簿から NO番の人のデータを削除するには

```
K ^MBM(NO)
```

で十分というわけです。さて



```
>K ^GLOBAL↵
```

としたらどうなるでしょうか？　^GLOBAL というグローバル変数がすべて削除されてしまいます。確かめてください。%GL を用いると

```
··グローバルが存在しません
```

というメッセージが返ってくるはずです。

しかし，これはとても恐ろしいことです。

```
K ^MBM
```

という命令を実行してしまうと，職員名簿に何千人，何万人のデータが入っていてもすべて削除されてしまいます。

MUMPS では KILL 命令は十分注意して使用しなくてはなりません。

- 変数をKILLすると，その「子供」に当たる変数（「下位節」）はすべて削除される

4,データを削除するルーチン

KILL 命令を利用して，登録されている人を削除するルーチンです。

list 7-1

```
MB027   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1985.3.15;ｻｸｼﾞｮ ﾙｰﾁﾝ
        ;
KAISHI  F K=1:1:1000 D BANGO I NO="*" Q
        W !,"ｻｸｼﾞｮ ｵﾜﾘ"
        Q
BANGO   R !,"ｻｸｼﾞｮ ｽﾙ ﾊﾞﾝｺﾞｳ ? ",NO
        I NO="*" Q
HYOJI   W !,"ﾅﾏI ..... = ",^MBM(NO,0)
        W !,"ｾｲﾍﾞﾂ .... = ",^MBM(NO,1)
        W !,"ﾈﾝﾚｲ..... = ",^MBM(NO,2)
        W !,"ｼｮｿﾞｸ･ﾌﾞｼｮ = ",^MBM(NO,3)
KAKUNIN R !,"ﾎﾝﾄｳ ﾆ ｻｸｼﾞｮ ｼﾏｽｶ ? ",KOTAE
SAKUJO  I KOTAE="Y" K ^MBM(NO) W !,"ｻｸｼﾞｮ ｼﾏｼﾀ"
        Q
```

このルーチンの働きは次の通りです。

①行 KAISHI の FOR 命令と DO 命令で，行 BANGO 以降を繰り返し呼び出します。呼び出した結果，変数 NO に『＊』記号が１文字だけセットされてくると，そこでこの繰り返しは終了します。

②行 BANGO は，行 KAISHI の DO 命令で呼び出されると，READ 命令で削除すべき登録番号を質問します。あなたが入力した番号は変数 NO にセットされます。

③もし変数 NO の値が『＊』記号1文字だけなら QUIT 命令を実行し，行 KAISHI に戻ってしまいます。

④変数 NO の値が『＊』記号でない場合は，行 HYOJI からの4行で登録されているデータを表示します。

⑤行 KAKUNIN では，本当に削除するのかどうか念をおす質問をします。

⑥質問にYと答えると，行 SAKUJO でK ^MBM(NO) という KILL 命令を実行し，変数を削除します。

削除する前に，職員名簿の全データを MB015 で出力してみましょう。

```
>D ^MB015↵

ｻｲｺﾞ ﾉ ﾊﾞﾝｺﾞｳ ? 7↵
1     ｲﾜｻｷ ﾋﾛﾐ          F     26    ｺｳﾎｳｼｼﾂ
2     ﾔﾏﾀﾞ ﾀﾛｳ          M     19    ﾍﾝｼｭｳﾌﾞ
3     ｼﾏ ﾖｼﾅﾘ           M     30    ｲﾑｼﾂ
4     ﾋﾗﾉ ｼﾞｭﾝｺ         F     27    ﾋｼｮｶ
5     ｷﾀﾑﾗ ｹﾝ           M     36    ｾｯｹｲﾌﾞ
6     ﾊｼﾓﾄ ｱﾂｺ          F     22    ﾀﾞｲ2 ｴｲｷﾞｮｳﾌﾞ
7     ﾋﾗﾀ ﾐﾎｺ           F     22    ｹｲﾘﾌﾞ
*** ｵﾜﾘ ***
>
```

いまこのように7人のデータが登録されています。プログラム ^%GL で調べると次のように表示されるはずです（MB027)。

```
>D ^%GL↵

グローバル　リスト　ユーティリティ

グローバルの選択: ^MBM↵

出力装置 <0> ↵
スクロールさせますか? <N> ↵
 ^MBM("1","0")="ｲﾜｻｷ ﾋﾛﾐ"
 ^MBM("1","1")="F"
 ^MBM("1","2")="26"
```

```
^MBM("1","3")="ｺｳﾎｳｼﾂ"
^MBM("2","0")="ﾔﾏﾀﾞ ﾀﾛｳ"
^MBM("2","1")="M"
^MBM("2","2")="19"
^MBM("2","3")="ﾍﾝｼｭｳﾌﾞ"
^MBM("3","0")="ｼﾏ ﾖｼﾅﾘ"
^MBM("3","1")="M"
^MBM("3","2")="30"
^MBM("3","3")="ｲﾑｼﾂ"
^MBM("4","0")="ﾋﾗﾉ ｼﾞｭﾝｺ"
^MBM("4","1")="F"
^MBM("4","2")="27"
^MBM("4","3")="ﾋｼｮｶ"
^MBM("5","0")="ｷﾀﾑﾗ ｹﾝ"
^MBM("5","1")="M"
^MBM("5","2")="36"
^MBM("5","3")="ｾｯｹｲﾌﾞ"
^MBM("6","0")="ﾊｼﾓﾄ ｱﾂｺ"
^MBM("6","1")="F"
^MBM("6","2")="22"
^MBM("6","3")="ﾀﾞｲ2 ｴｲｷﾞｮｳﾌﾞ"
^MBM("7","0")="ﾋﾗﾀ ﾐﾎｺ"
^MBM("7","1")="F"
^MBM("7","2")="22"
^MBM("7","3")="ｹｲﾘﾌﾞ"

グローバルの選択: ↵          ：ここで何も入力せずにリターンキーを押す
>                                と%GLルーチンが終了します
```

さてルーチン MB027 を用いて，3番の人のデータを削除します。

```
>D ^MB027↵

ｻｸｼﾞｮ ｽﾙ ﾊﾞﾝｺﾞｳ ? 3↵
ﾅﾏｴ .....= ｼﾏ ﾖｼﾅﾘ
ｾｲﾍﾞﾂ .....= M
ﾈﾝﾚｲ .....= 30
ﾃﾞﾝﾜ ﾊﾞﾝｺﾞｳ = ｲﾑｼﾂ
ﾎﾝﾄｳ ｺ ｻｸｼﾞｮ ｼﾏｽｶ ? Y↵
ｻｸｼﾞｮ ｼﾏｼﾀ
ｻｸｼﾞｮ ｽﾙ ﾊﾞﾝｺﾞｳ ? *↵          ：'*' を入力すると削除業務が終わ
ｻｸｼﾞｮ ｵﾜﾘ                          ります
>
```

^%GL でデータをリストしてみましょう。

```
>D ^%GL↵

グローバル　リスト　ユーティリティ

グローバルの選択: ^MBM↵

出力装置.<0> ↵
スクロールさせますか? <N> ↵
^MBM("1","0")="ｲﾜｻｷ ﾋﾛﾐ"
^MBM("1","1")="F"
^MBM("1","2")="26"
^MBM("1","3")="ｺｳﾎｳｼｯ"
^MBM("2","0")="ﾔﾏﾀﾞ ﾀﾛｳ"
^MBM("2","1")="M"
^MBM("2","2")="19"
^MBM("2","3")="ﾍﾝｼｭｳﾌﾞ"                    :^MBM(3) とその下位節（子供）が
^MBM("4","0")="ﾋﾗﾉ ｼﾞｭﾝｺ"                    削除されているのが分かります
^MBM("4","1")="F"
^MBM("4","2")="27"
^MBM("4","3")="ﾋｼｮｶ"
^MBM("5","0")="ｷﾀﾑﾗ ｹﾝ"
^MBM("5","1")="M"
^MBM("5","2")="36"
^MBM("5","3")="ｾｯｹｲﾌﾞ"
^MBM("6","0")="ﾊｼﾓﾄ ｱﾂｺ"
^MBM("6","1")="F"
^MBM("6","2")="22"
^MBM("6","3")="ﾀﾞｲ2 ｴｲｷﾞｮｳﾌﾞ"
^MBM("7","0")="ﾋﾗﾀ ﾐﾎｺ"
^MBM("7","1")="F"
^MBM("7","2")="22"
^MBM("7","3")="ｹｲﾘﾌﾞ"

グローバルの選択: ↵
>
```

たしかにグローバル変数 ^MBM(3) とその下位節（「子供」）が削除され，登録番号３番の人のデータを入れた変数がこのリストに入っていません。

この「親」が削除されると自動的に「子供」も消えるということの，データ処理における強力さを想像してみてください。

5,データの削除による問題点

削除ができると今度はデータのリストを作るルーチンに支障が生じてきます。削除されてしまったデータは出力できない（ない袖は振れない）ということで，〈UNDEF〉 エラーになってしまうのです。

ルーチン MB015 を使ってデータをリストしてみましょう。

```
>D ^MB015↵

ｻｲｺﾞ ﾉ ﾊﾞﾝｺﾞｳ ﾊ ? 7↵
1    ｲﾜｻｷ ﾋﾛﾐ          F     26    ｺｳﾎｳｼｼﾂ
2    ﾔﾏﾀﾞ ﾀﾛｳ          M     19    ﾍﾝｼｭｳﾌﾞ
3
<UNDEF> 定義されていない変数が参照されました
 1) +4^MB015:: F J=1:1:NN W !,J,?5,^MBM(J,0),?20,^MBM(J,1),?25,^MBM(J,2...
 0) *XECUTE*^MB015:: D ^MB015
<UNDEF>+4^MB015:::4:1 F J=1:1:NN W !,J,?5,^MBM(J,0),?20,^MBM(J,1),?25,^...
```

となり 〈UNDEF〉 エラーが発生してルーチンの実行は止まってしまいます。なぜでしょう？ このルーチンのリストを見てください。

list 7-2

```
MB015   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1985.2.19;ﾃﾞｰﾀ ﾉ ﾘｽﾄｱｯﾌﾟ
        ;
HAJIME  R !,"ｻｲｺﾞ ﾉ ﾊﾞﾝｺﾞｳ ﾊ ? ",NN
        F J=1:1:NN W !,J,?5,^MBM(J,0),?20,^MBM(J,1),?25,^MBM(J,2),?30,^MBM(J,3)
        W !,"*** ｵﾜﾘ ***"
        Q
```

FOR 命令によって変数 J の値は 1 から 1 ずつ増えて，そのたびに ^MBM(J,0)，^MBM(J,1)，^MBM(J,2)，^MBM(J,3) という 4 つのグローバル変数を表示します。

ところが，今，3 番のデータが削除されているので，^MBM(3,0) というグローバル変数が存在しません。J=3 になったとき，この変数の内容を表示しようとして，それが見当たらないということで 〈UNDEF〉 エラーが生じたのです。

- 存在しない変数の値を参照しようとすると〈UNDEF〉というエラーになる

6,データが存在するかどうかを調べる

$DATA 関数

あるデータが存在するかどうかを調べるために，MUMPS には $DATA 関数という機能が用意されています（「関数」という言葉については「関数について」参照)。この関数を用いて，変数の値を表示する前に，それが存在するか否かをチェックすることができます。

$DATA（ダラーデータと呼んでください）関数について解説します。$DATA 関数は

$D(変数)

という形式で，WRITE 命令，SET 命令，IF 命令などと組み合わせて用いることができます。

いま「職員名簿」の登録番号4番，5番にはデータが登録されていますが，3番や9番にはデータが登録されていません。次のようにタイプしてください。

```
>W $D(^MBM(4))↵
10                    ：10に注意！
>W $D(^MBM(5))↵
10                    ：これも10です
>W $D(^MBM(3))↵
0                     ：0に注意！
>W $D(^MBM(9))↵
0                     ：これも0です
>
```

$DATA関数のカッコ（ ）の中に，^MBM(4) や ^MBM(5) を指定すると10という値が表示され，^MBM(3) や ^MBM(9) を指定すると0という値が表示されました。

データが登録されている番号を添え字とすると10が，データが登録されていない番号を添え字とすると0が表示されるわけです。

この機能を利用して，〈UNDEF〉エラーを回避するように，データのリストを書き出すルーチン MB015 を作りなおしましょう。

list 7-3

```
MB028   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1985.2.19;ﾃﾞｰﾀ ﾉ ﾘｽﾄｱｯﾌﾟ
        ;
HAJIME  R !,"ｻｲｺﾞ ﾉ ﾊﾞﾝｺﾞｳ ﾊ ? ",NN
        F J=1:1:NN I $D(^MBM(J))=10 W !,J,?5,^MBM(J,0),?20,^MBM(J,1),?25,^MBM(J,
    -->2),?30,^MBM(J,3)  ①
        W !,"*** ｵﾜﾘ ***"
        Q
```

ここに注目!!

実行してみます。エラーは出ませんね。

```
>D ^MB028↵

ｻｲｺﾞ ﾉ ﾊﾞﾝｺﾞｳ ﾊ ? 7↵
1     ｲﾜｻｷ ﾋﾛﾐ          F     26    ｺｳﾎｳｼｼﾂ
2     ﾔﾏﾀﾞ ﾀﾛｳ          M     19    ﾍﾝｼｭｳﾌﾞ
4     ﾋﾗﾉ ｼﾞｭﾝｺ         F     27    ﾋｼｮｶ
5     ｷﾀﾑﾗ ｹﾝ           M     36    ｾｯｹｲﾌﾞ
6     ﾊｼﾓﾄ ｱﾂｺ          F     22    ﾀﾞｲ2 ｴｲｷﾞｮｳﾌﾞ
7     ﾋﾗﾀ ﾐﾎｺ           F     22    ｹｲﾘﾌﾞ
*** ｵﾜﾘ ***
>
```

：3番がスキップされています

このルーチンの4行目の，I $D 以下＝10までの IF 命令と $DATA 関数に注意してください（下線部分です）。

FOR 命令は変数Jの値を1ずつ増やして，そのたびにこの IF 命令を実行します。

IF 命令は各Jの値について，$D(^MBM(J))＝10 が正しいかどうかを調べます。そして，これが正しければ次の WRITE 命令を実行し，そうでなければ WRITE 命令を実行しません。したがって WRITE 命令で表示するときには決してエラーになりません。

◎ 関数について

「関数」という言葉は数学の用語なので，少々難しく感じるかもしれません。これは，英語の function（機能，働き，職務）という言葉の訳語で，数学とは関係ないのですが，MUMPS では慣例で「関数」と呼んでいます。

「関数」とは一般に，データとか名前といった'何か'をもとにして，ある決められた手順にしたがって'値'を作り出すものです。

たとえば，$DATA 関数は変数の名前をもとに0, 1, 10, 11という 4 つの数値のうちどれかを返します。

$LENGTH 関数はデータをもとにそのデータに含まれる文字の数を算出します（ file 2 参照)。

「関数」は ' 値 ' を算出するので，それを表示したり，計算に用いることができます。

```
>W $D(X)↵
1
>S A=$L("ABC") W A↵
3
>
```

これは $LENGTH 関数で文字の数を算出します

MUMPS の「関数」の特徴は，みな先頭に「$」記号をもち，そのあとにそれぞれ固有な名前（の頭文字)，さらにその右にカッコ( )を持っています。

```
$D(X)  $O(A(5))  $L(X)
```

などで，この MUMPS の慣例に従うと，数学の関数 sin x は，$SIN(X) と命令することができ，log x は $LOG(X) と命名することができます（このような関数は標準 MUMPS にはないので注意！— SP-MUMPS の数値関数については file 2 参照)。

- 職員名簿グローバル変数の登録番号 J 番のデータがあると $D(^MBM (J)) の値は10である
- 職員名簿グローバル変数の登録番号 J 番のデータが存在しないと $D (^MBM (J)) の値は 0 である

7, $DATA 関数とは

$DATA 関数についてもう少し解説しましょう。前節で述べた通り，$DATA 関数は

```
$D(変数)        ：変数はグローバル変数でもローカル変数でもよい
```

という形式で，その変数が存在するか否か，そしてその変数が存在するとしたらどのようなものかを表す値を返してくれます。

値を返すというのは，その値を WRITE 命令で表示したり（W $D(^MBM

(3))), IF 命令で参照したり（I $D（^MBM(J))＝10)することができるということです。

そして存在しないときは0，存在するときは10という値を返すことが分かりました。

しかしちょっと待ってください ^MBM(1)，^MBM(2),……，^MBM(7)という変数に値があるわけではありません。これらの変数を WRITE しようとするとエラーになってしまいます。たとえば

```
>W ^MBM(4)↵

<UNDEF> 定義されていない変数が参照されました
 0) *XECUTE*:: W ^MBM(4)
<UNDEF>*XECUTE*:::4:1 W ^MBM(4)
>
```

となります。これらは，例の「仮想的な親」でした。実際には4番の人のデータは，次のような

```
^MBM(4,0)
^MBM(4,1)
^MBM(4,2)
^MBM(4,3)
```

4つの変数に入っているからです。

```
>W ^MBM(4,0)↵
ﾋﾗﾉ ｼﾞｭﾝｺ
>W ^MBM(4,1)↵
F
>W ^MBM(4,2)↵
27
>W ^MBM(4,3)↵
ﾋｼｮｶ
>
```

$D(^MBM(4)) の値が10というのは，^MBM(4) それ自身にはデータはないが，下位節（「子供」）があるという意味なのです。

ではこの「子供」達の $DATA の値はどうなるでしょう？

試してみましょう。

```
>W $D(^MBM(4,0))
1                              ：1が表示されました
>W $D(^MBM(4,1))
1
>W $D(^MBM(4,2))
1
>W $D(^MBM(4,3))
1
>
```

このようにすべて1が表示されます。

こんどは存在しない「子供」を指定してみます。

```
>W $D(^MBM(4,12))
0                              ：0です！
>W $D(^MBM(4,98))
0
>
```

やはり0が返されます。

さらに，$DATA は11という値を返すこともあります。これは「自分にもデータがあり，かつ，子供もある」という変数であることを示す値です。この職員名簿の例では不適当なので次の例を見てください（ここではローカル変数を例にします)。

```
>K DA          ：変数DAをその下位節とともに削除

>S DA(2)="ｷｻﾗｷﾞ"

>S DA(2,14)="ﾊﾞﾚﾝﾀｲﾝ ﾃﾞｰ"          ：変数DAの下位節を新たにセット

>S DA(3)="ﾔﾖｲ"

>S DA(3,3)="ﾐﾐ ﾉ ﾋ"

>S DA(3,14)="ﾎﾜｲﾄ ﾃﾞｰ"

>S DA(3,18)="ﾀﾞﾚｶ ﾉ ﾀﾝｼﾞｮｳﾋﾞ"

>S DA(8,6)="ﾋﾛｼﾏ ｹﾞﾝﾊﾞｸ ｷﾈﾝﾋﾞ"
```

DA(2) と DA(3) という変数は，それ自身が ' キサラギ ' ，' ヤヨイ ' と

いう値を持っていますが，さらに，DA(2,14)，DA(3,18) などの下位節（「子供」）もあります。したがって

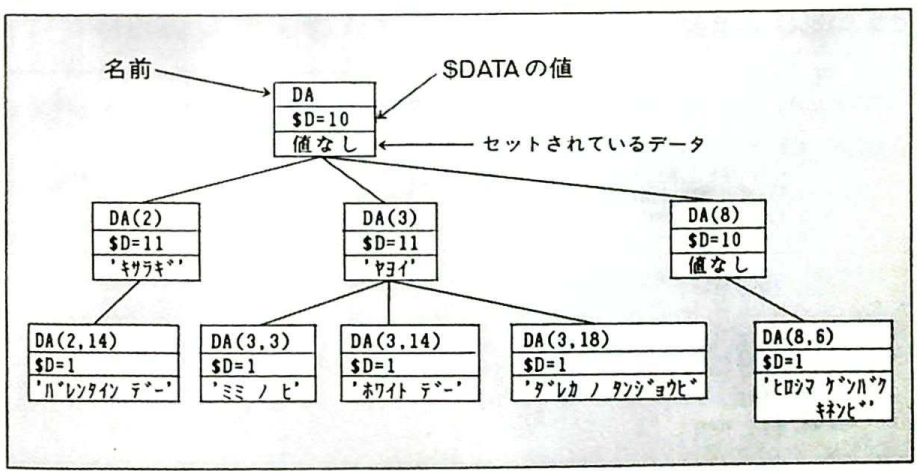
```
>W $D(DA(2))
11                    ：11です!!
>W $D(DA(3))
11                    ：これも11です
>
```

となります。いろいろな変数について実際に確かめてください。

```
>W $D(DA(3,14))       ：「ホワイトデー」は定義されています
1
>W $D(DA(12))         ：「師走」はセットしてありません
0
>W $D(DA(6,4))        ：「虫歯予防デー」もセットしてありません
0
>W $D(DA(8))          ：DA(8,6)は定義しましたが，DA(8)はセット
10                      してありません。したがって$DATAの値は10
>W $D(DA)             ：添え字のない変数DAにとって，DA(2)，DA
10                      (3)，DA(2,14)などはすべて下位節(子)にな
>                       ります。DA自身は定義されていないので$D
                        (DA)＝10です
```

図解すると fig 7-2 の通りです。ここに書いてない変数はみな $DATA の値が0になります。

fig 7-2　$DATA相関図



$DATA の値をまとめると次の通りです。

指定された変数がまったく存在しないとき…… 0(00)
指定された変数は存在するが下位節は存在しないとき…… 1(01)
指定された変数は存在しないが下位節は存在するとき……10(10)
指定された変数も下位節も存在するとき……11(11)

0と1という値をそれぞれ00と01として考えると（　）内のようになります。こうすると，10の位は下位節（「子供」）の有無を，1の位はその変数自身の有無を示していると解釈することができます。

- $DATA関数を用いて変数とその下位節の有無を調べることができる
  $D (変数) の値：
  指定された変数がまったく存在しないとき…… 0
  指定された変数は存在するが下位節は存在しないとき… 1
  指定された変数は存在しないが下位節は存在するとき…10
  指定された変数も下位節も存在するとき……11
- $DATA関数は計算式，判断式などの中で用いることができる

8,条件によりデータリストを改造する

$DATA 関数を用いて男性のみのデータリストを作ってみましょう。ルーチン MB017 の改造です。

list 7-4

```
MB029   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1985.3.19;ﾀﾞﾝｾｲ ﾉﾐﾉ ﾃﾞｰﾀ ﾉ ﾘｽﾄｱｯﾌﾟ($DATA ﾉ ﾘﾖｳ)
        ;
HAJIME  R !,"ｻｲｺﾞ ﾉ ﾊﾟﾝｺﾟｳ ﾊ ? ",NN
        F J=1:1:NN I $D(^MBM(J))=10 I ^MBM(J,1)="M" W !,J,?5,^MBM(J,0),?20,^MBM(
     -->J,1),?25,^MBM(J,2),?30,^MBM(J,3)
        W !,"*** ｵﾜﾘ ***"
        Q
```

注目するのはここ

```
>D ^MB029

ｻｲｺﾞ ﾉ ﾊﾟﾝｺﾟｳ ﾊ ? 7
2     ﾔﾏﾀﾞ ﾀﾛｳ          M     19    ﾍﾝｼｭｳﾌﾞ
5     ｷﾀﾑﾗ ｹﾝ            M     36    ｾｯｹｲﾌﾞ
*** ｵﾜﾘ ***
>
```

行 HAJIME+1 の下線の引いてある IF 命令に注目してください。ふたつの IF 命令が重なっています。

この部分の解釈の仕方を少し詳しく見てみましょう。

まず，FOR 命令によって与えられた J について，左側の（FOR 命令のすぐ右にある）IF 命令

```
I $D(^MBM(J))=10
```

が実行されます。もし J 番の人が登録されていないと，$D(^MBM(J))の値は 0 ですからこの IF 命令の条件は正しくないので（「偽」なので），これより右にある命令はすべて実行されません。

もし J 番の人が登録されていれば $D(^MBM(J))＝10は正しいので，さらに右側の IF 命令

```
I ^MBM(J,1)="M"
```

へ実行が移ります。

次にこの条件 ^MBM(J,1)＝"M" が正しければ（J 番の人が男性なら）その右にある WRITE 命令が実行され，データが表示されます。しかし，女性ならこの条件は正しくないので WRITE 命令は実行されません。

結局 J 番の人が登録されていて，かつ男性の場合にのみデータが表示されることになります。

さて，ここでこのふたつの IF 命令の順序を入れ替えたらどうなるでしょう。

```
I ^MBM(J,1)="M" I $D(^MBM(J))=10 W !,J,?5,^MBM(J,0),?20,…(省略)
```

この場合，まず左側の IF 命令

```
I ^MBM(J,1)="M"
```

が実行されますが，この命令は変数 ^MBM(J,1) を参照しています。

そこでもし J 番の人が，登録されていないと，この変数は「未定義」である (SET されていない) ため，エラーになってしまいます。試してみましょう。

list 7-5

```
MB030   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1985.3.19;ﾀﾞﾝｾｲ ﾉﾐﾉ ﾃﾞｰﾀ ﾉ ﾘｽﾄｱｯﾌﾟ($DATA ﾉ ﾘﾖｳ)
        ;
HAJIME  R !,"ｻｲｺﾞ ﾉ ﾊﾞﾝｺﾞｳ ﾊ ? ",NN
        F J=1:1:NN I ^MBM(J,1)="M" I $D(^MBM(J))=10 W !,J,?5,^MBM(J,0),?20,^MBM(
    -->J,1),?25,^MBM(J,2),?30,^MBM(J,3)
        W !,"*** ｵﾜﾘ ***"
        Q
```

```
>D ^MB030↵

ｻｲｺﾞ ﾉ ﾊﾟﾝｺﾞｳ ﾊ ? 7↵
2    ﾔﾏﾀﾞ ﾀﾛｳ        M     19    ﾍﾝｼｭｳﾌﾞ ←
<UNDEF> 定義されていない変数が参照されました
 1) +4^MB030:: F J=1:1:NN I ^MBM(J,1)="M" I $D(^MBM(J))=10 W !,J,?5,...
 0) *XECUTE*^MB030:: D ^MB030
<UNDEF>+4^MB030:::4:1 F J=1:1:NN I ^MBM(J,1)="M" I $D(^MBM(J))=10 W !...
>W J↵
3 ←
>
```

J＝2までは実行されています

エラーが出たときJ＝3です
^MBM(3,1)が未定義なのです

この例ではJ=3のとき^MBM(J,1)（つまり^MBM(3,1)）が定義されていなかったためエラーになってしまいました。右側のIF命令で$DATA関数を使用した意味がまったくなかったことになります。このようにIFの条件付けを重ねるときは，その順序が重要な意味を持つことがあるので注意してください。

ところで，IF命令も複数の引数を持つことができます。

```
I $D(^MBM(J))=10 I ^MBM(J,1)="M"
```

というふたつのIF命令は，ふたつの引数を持つIF命令

```
I $D(^MBM(J))=10,^MBM(J,1)="M"
```

で代用することができます。この場合も左側の条件が満たされないとき（「偽」のとき）IF命令の実行はそこで終了してしまい，右側の条件式は無視されます。したがって，このようにひとつのIF命令にまとめてしまっても，なんら差し支えありません。

またこの条件を

```
I ^MBM(J,1)="M",$D(^MBM(J))=10
```

と逆にしてはいけないこともIF命令をふたつ重ねた場合と同じです。

最後にひとつのIF命令にまとめた，男性をリストするルーチン例を示します。

list 7-6

```
MB031   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1985.3.19;ﾀﾞﾝｾｲ ﾉﾐﾉ ﾃﾞｰﾀ ﾉ ﾘｽﾄｱｯﾌﾟ($DATA ﾉ ﾘﾖｳ)
        ;
HAJIME  R !,"ｻｲｺﾞ ﾉ ﾊﾟﾝｺﾞｳ ﾊ ? ",NN
        F J=1:1:NN I $D(^MBM(J))=10,^MBM(J,1)="M" W !,J,?5,^MBM(J,0),?20,^MBM(J,
     -->1),?25,^MBM(J,2),?30,^MBM(J,3)
        W !,"*** ｵﾜﾘ ***"
        Q
```

実行してみます。

```
>D ^MB031↵

ｻｲｺﾞ ﾉ ﾊﾞﾝｺﾞｳ ﾊ ? 7↵
2     ﾔﾏﾀﾞ ﾀﾛｳ          M      19     ﾍﾝｼｭｳﾌﾞ
5     ｷﾀﾑﾗ ｹﾝ            M      36     ｾｯｹｲﾌﾞ
*** ｵﾜﾘ ***
>
```

- IF 命令もふたつ以上の引数を持つことができ，それはひとつの引数を持つ IF 命令を複数並べた場合と等価である
- IF 命令の引数が複数あるときは，左からひとつずつ条件判断される。そして条件が「偽」のときはそこで IF 命令の実行が終わり，それより右にある引数や命令はすべて無視される
- 「変数 XX が存在して，かつ，その値が2であれば」という条件は

  I $D(XX)=1, XX=2

  という IF 命令で表される
- 'I $D(XX)=1,XX=2' と 'I XX=2,$D(XX)=1' とは等価ではなく，後者の場合 XX が未定義のときにエラーとなる

Chapter 8

データの兄弟をさがす

前章で学習した $DATA 関数は，いわば変数の親子関係を調べる機能でした。この章では，変数の「兄弟」を調べる機能 $ORDER 関数について解説します。この機能は，大量に蓄えたデータを調べたり，データを並べ替えるときに便利な関数です。

1,登録番号が飛び離れている場合の問題

もう一度すべての人のデータをリストするルーチンを検討しましょう。

list 8-1

```
MB028   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1985.3.17;ﾃﾞｰﾀ ﾉ ﾘｽﾄｱｯﾌﾟ
        ;
HAJIME  R !,"ｻｲｺﾞ ﾉ ﾊﾞﾝｺﾞｳ ﾊ ? ",NN
        F J=1:1:NN I $D(^MBM(J))=10 W !,J,?5,^MBM(J,0),?20,^MBM(J,1),?25,^MBM(J,
    -->2),?30,^MBM(J,3)
        W !,"*** ｵﾜﾘ ***"
        Q
```

このルーチンの欠点はふたつあります。

ひとつは登録されている人の番号の最大値（NN）が分からないと使えないということです。

もうひとつは，1番から NN 番までひとつずつ順番に $DATA でデータがあるか否かを調べていることです。

現実のデータでは，登録番号が連続していないことが普通です。たとえば年

■notes

1) MUMPSではBASICなど他の多くのシステムとは異なり，変数の添え字の値が連続している必要はありません。とびとびの値でもMUMPSは必要最小限の記憶領域を使います。したがって，X(1)とX(900)しかセットされていないとき，X(2)からX(899)までの領域を無駄に使うことになりません。

月日でデータを整理することがあります。このようなデータでは変数の添え字は当然連続していません。[1)]

```
^SHIIRE(860131) ･･･ 1986 年 1 月 10 日の仕入れ金額合計
^SHIIRE(860201) ･･･ 1986 年 2 月 1 日の仕入れ金額合計

 860132 から 860200 までの添え字は
 論理的にあり得ません

 860158 ---> 1986 年 1 月 58 日  ???
```

登録されていない番号を無駄に調べることは，いくらコンピュータが高速に仕事をするといっても時間がかかります。とても実用的とはいえません。

このような問題を解決するためにはいくつかの機能を新しく学ばなくてはなりません。

2,空文字列の発見

まず,「空文字列」という概念を理解してください。「空文字列」とは文字のないデータ，言い換えると０文字のデータのことです。次の例をみてください。

```
"ABC"     は，3 文字のデータです
"AB"      は，2 文字のデータです
"A"       は，1 文字のデータです
```

そして

```
""        は，  0 文字のデータです
```

最後の０文字のデータを「空文字列」と呼んでいます。

変数に空文字列をセットするには

```
>S AA=""↵
```

とします。

この変数の値を調べてみましょう。

```
>W AA↵
                ：何も表示されません
>
```

何も表示されません。$DATA 関数を使って調べます。

```
>W $D(AA)↵
1
>
```

1が表示されるということは変数 AA に何かの値がセットされているということです。「文字のない文字列」は透明人間のようなものです。少し考えをすすめるために工夫しましょう。

```
>W "*",AA,"*"↵
**
>
```

ふたつの『＊』記号が並んで表示されました。変数 AA は表示されたのでしょうか？

いったん変数 AA を KILL してみましょう。

```
>K AA↵                              ：変数を削除しました

>W "*",AA,"*"↵
*                                   ：AAの左の「＊」記号だけが表示されました
<UNDEF> 定義されていない変数が参照されました
 0) *XECUTE*:: W "*",AA,"*"   ：エラーです
<UNDEF>*XECUTE*:::4:0 W "*",AA,"*"
>
```

〈UNDEF〉 エラーになってしまいました。KILL 命令で削除した変数を表示しようとしたのですから当然です。$DATA 関数はどうなっているのでしょう。

```
>W $D(AA)↵
0
>
```

0ということは変数 AA がまったく存在しないということです。

もういちど AA をセットしてから表示しましょう。

```
>S AA=""↵

>W "*",AA,"*"↵
**
>
```

やはりこんどはエラーにはなりませんでした。

変数 AA は確かに存在しているし，その値は表示されているのですが，「文字のない文字列」であるため見ることができないのです。

次の命令は変数 AA の値が空文字列ならば 'シキ ソク ゼ クウ' と表示します。

```
>I AA="" W "ｼｷ ｿｸ ｾﾞ ｸｳ"↵
ｼｷ ｿｸ ｾﾞ ｸｳ
>
```

となります。

また，READ 命令で何もデータをタイプせずに，↵キーのみを打つと空文字列が入力されます。

```
>R "ﾃﾞｰﾀ = ",X I X="" W !,"ｶﾗ ﾓｼﾞﾚﾂ"↵
ﾃﾞｰﾀ = ↵                          :↵キーのみを打ちました
ｶﾗ ﾓｼﾞﾚﾂ                          :'カラモジレツ'と表示されました
>
```

何も難しいことではありませんが，数学の「0」と同じように，MUMPS ではこの空文字列が重要な概念のひとつなので十分に理解してください。

昔、インド人は"0"を発見しました。
今、あなたは"空文字列"を発見しました。

- 0 文字の文字列を空文字列という
- 空文字列を表すには，「""」と表す
- 空文字列をセットするには，SX=""とする
- READ命令で，データを入力せずに，ただ↵キーのみを打つと空文字列を入力したことになる

3,次の添え字を捜す

$ORDER 関数

次に，$ORDER 関数という機能について解説します。

$ORDER は変数の兄弟を調べる関数で，これを利用するとすぐ上の「兄」

にあたる変数の添え字の値を知ることができます。

たとえば，まず変数 CD を次のようにセットしておきます。

```
>K CD↵                          ：変数CDとその下位節(子供達)を削除
>S CD(17)="ｳｼｼﾞﾏ"↵
>S CD(29)="ｽｽﾞｷ"↵
>S CD(30)="ｶｸ"↵                 ― 改めて変数CDの下位節をセット
>S CD(34)="ﾌﾏﾂ"↵
>S CD(41)="ﾔｻﾞﾜ"↵
>
```

ここで，$ORDER 関数の値を WRITE しましょう。$ORDER は $O と省略されます。

```
>W $O(CD(17))↵
29                              ：17の次は29
>W $O(CD(29))↵
30                              ：29の次は30
>W $O(CD(30))↵
34                              ：30の次は34
>W $O(CD(34))↵
41                              ：34の次は41
>
```

このように $ORDER 関数で指定した変数の次にあたる変数の添え字が表示されます。CD(29) の次の変数は CD(30) なので，$O(CD(29)) の値は30です。

ところで，$O (CD(41)) はどうなるのでしょう？

```
>W $O(CD(41))↵

>
```

何も出てきません。もういちど，今度は『＊』記号を一緒に表示します。

```
>W "*",$O(CD(41)),"*"
**
>
```

『＊』と『＊』の間には何も文字がありません。

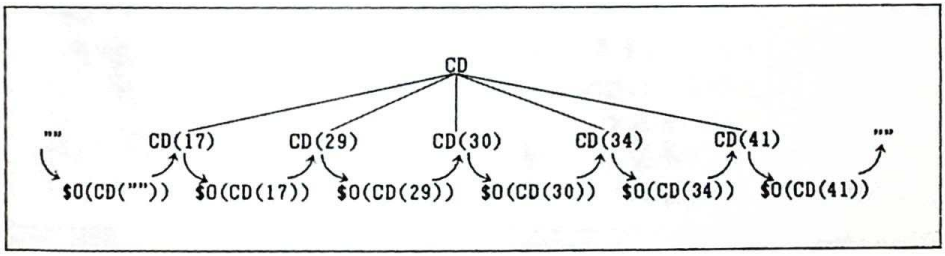
$ORDER 関数は，空文字列を返してきたのです。$ORDER 関数は指定された変数より大きい添え字の値を持つ変数が存在しないときこのように空文字列を返します。

では，一番小さな添え字の値を調べるにはどうしたらよいでしょう。これにも空文字列が活用されます。$O(CD("")) とすると最小の添え字が返されます。

```
>W $O(CD(""))
17
>
```

$ORDER 関数は空文字列に始まり空文字列に終わるループを描いて添え字を順番にさがしてくれます。

fig 8-1 SORDER関数のループ



職員名簿の ^MBM というグローバル変数についても同じことです。

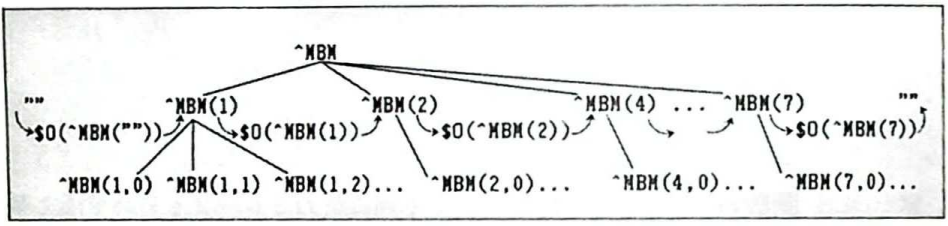
```
>W $O(^MBM(""))
1                          ：最初の添え字です
>W $O(^MBM(1))
2
>W $O(^MBM(2))
4                          ：3がスキップされていることに注意
>W $O(^MBM(4))
5
>W $O(^MBM(5))
6
>W $O(^MBM(6))
```

```
7
>W $O(^MBM(7))↵
                        :空文字列です
>
```

前章で3番の人のデータ（^MBM(3)）を削除していたので，添え字3がスキップされています。また，$ORDER の中の最初の添え字は空文字列で，$ORDER が最後に返してくれる値は空文字列になっています。

これらの変数はすべて値を持たない「仮想的な親」であることにも注意してください。

fig 8-2　$ORDER関数とグローバル変数



添え字が複数の場合の $ORDER 関数の働きについては，Chapter 9 を参照してください。

- $ORDER関数は指定された変数の次に小さい添え字を返す
- 指定された変数より大きな添え字の変数がないときは，空文字列を返す
- 最小の添え字を捜すには$ORDER関数で添え字に空文字列を指定する

4,繰り返しの方法

さて FOR 命令と $ORDER 関数を組み合わせると，次のようなルーチンを作ることができます。

list 8-2

```
MB032   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1985.5.17;$ORDERｶﾝｽｳ ﾉ ﾚｲ,  FOR ﾒｲﾚｲ ﾆ ﾖﾙ ｸﾘｶｴｼ
        ;
        S X=""
        F J=1:1:7 S X=$O(^MBM(X)) W !,"*",X,"*"
        Q
```

このふたつの命令が繰り返される

```
>D MB032↵

*1*                    ：添え字の前後に『＊』記号を表示しています
*2*
*4*                    ：3はスキップされています
*5*
*6*
*7*
**                     ：空文字列です
>
```

このルーチンの内容をよく検討してみましょう。

FOR 命令は，その右にある命令を7回繰り返すことを指定します。ここでは

```
S X=$O(^MBM(X)) W !,"*",X,"*"
```

というふたつの命令が繰り返されます。最初に，変数Xには空文字列がセットされているので，この命令は

```
S X=$O(^MBM(""))  W !,"*",X,"*"          ：下線部分に注意！
```

と解釈されます。変数Xには新たに $O(^MBM("")) の値，すなわち1がセットされ，それが WRITE 命令で表示されます。

FOR 命令の働きで，またこの命令が繰り返されます。そのときは X=1 なので

```
S X=$O(^MBM(1))  W !,"*",X,"*"           ：下線部分に注意！
```

と解釈され，変数Xには新たに $O(^MBM(1)) の値，すなわち2がセットされ，それが WRITE 命令で表示されます。

以下同様に変数Xに，順番に4, 5, 6, 7がセットされ，最後に X=7 のとき，

```
S X=$O(^MBM(7)) W !,"*",X,"*"        ：今，7より大きい添え字はありません
```

と解釈され，変数Xには空文字列がセットされます。画面には『**』と表示されます。

そして合計7回繰り返されたので FOR 命令の制御がここで終了します。

- FOR命令と$ORDER関数を組み合わせて，添え字を順に調べることができる

5, FOR命令とQUIT命令の組み合わせ

しかし，このルーチンの問題は，FOR 命令で繰り返す回数を 7 回と固定していることです。

ここで Chapter 6 で学んだ FOR 命令と QUIT 命令の組み合わせを思い出してください。変数Xの値が空文字列になったら繰り返しを中止することができます。

list 8-3

```
MB033   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ";1985.5.17;$ORDER ｶﾝｽｳ ﾉ ﾚｲ, FOR ﾒｲﾚｲ ﾆ ﾖﾙ ｸﾘｶｴｼ
        ;
        S X=""
        F J=1:1:9999 S X=$O(^MBM(X)) W !,"*",X,"*" I X="" Q
        Q
```

このルーチンは，本来なら

```
S X=$O(^MBM(X)) W !,"*",X,"*"
```

を9999回繰り返すことになるのですが，この行の最後に

```
I X="" Q
```

という命令があるために，登録人数＋1回しか繰り返しません。

SX=$O(^MBM(X)) で次の登録番号を捜し，WRITE 命令でそれを表示します。そのあと IF 命令を実行します。もし変数 X の値が空文字列ならX=""という条件が正しいので QUIT 命令が実行されます。

QUIT 命令の左側に FOR があると，QUIT 命令はその FOR 命令による繰り返しを終了させてしまいます。したがって，X="" となったところでこの繰り返しが終了するのです。今の職員名簿の例では 6 人しか登録されていないので，6つの登録番号を表示し，最後に空文字列を表示して，結局7回繰り返すことになります。

- FOR命令と同一行で，かつそのFOR命令の右側にあるQUIT命令はFOR命令による繰り返しを終了させる

■6,終わらないFOR命令

前節のルーチンの

```
F J=1:1:9999 S X=$O(^MBM(X)) W !,"*",X,"*" I X="" Q
```

という行のうち FOR 命令の9999という値には意味がありません。この数値には十分に大きい値を選んでおけば良いのです。何万人ものデータを扱う場合にはもっと大きな値が必要になります。

FOR 命令のこの最終値をとくに指定しない方法もあります。

```
F J=1:1 S X=$O(^MBM(X)) W !,"*",X,"*" I X="" Q
```

終わりの値がない!!

これは，いつまでも終わらない繰り返し（無限ループ）です。IF 命令の条件 X="" が「真」になったときに QUIT 命令が実行されて，そこで繰り返しが終了します。さらに

```
F QQ=0:0 S X=$O(^MBM(X)) W !,"*",X,"*" I X="" Q
```

初期値　増分

とすることもできます。

この場合，変数 QQ は0から0ずつ増えるので，結局ずっと0のままです。それでも右側にある SET や WRITE 命令を繰り返すことには変わりありません。

FOR 命令の繰り返しを数える変数の値が0のままというのは，直観的に分かりにくいかもしれません。しかし，これも MUMPS のプログラムのひとつの型として覚えてください。

これから，「繰り返す」ことに意味があって，繰り返す「回数」にはあまり意味がない場合には，この

```
F QQ=0:0 繰り返す命令
```

という形式を採用します。ルーチン MB033 を書き直してみましょう。

最終値を指定しない FOR 命令は，QUIT 命令がないと決して終了しないので注意してください。終わらない FOR 命令も試しましょう。

list 8-4

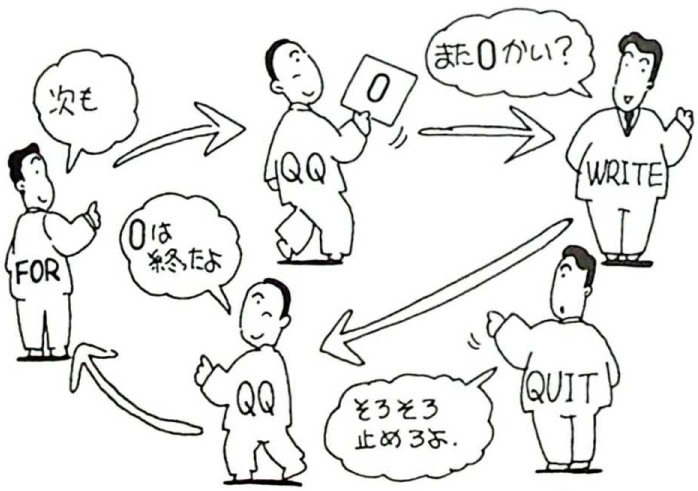
```
MB034   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1985.5.17;$ORDER ｶﾝｽｳ ﾉ ﾚｲ, FOR ﾒｲﾚｲ ﾆ ﾖﾙ ｸﾘｶｴｼ
        ;
        S X=""
        F QQ=0:0 S X=$O(^MBM(X)) W !,"*",X,"*" I X="" Q
        Q
```

QQはいつまでたっても0のままです

list 8-5

```
MB035   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1985.5.17;$ORDER ｶﾝｽｳ ﾉ ﾚｲ, FOR ﾒｲﾚｲ ﾆ ﾖﾙ ｸﾘｶｴｼ
        ;
        S X=""
        F QQ=0:0 S X=$O(^MBM(X)) W !,"*",X,"*"     ：QUITなし！
        Q
```

```
>D MB035↵
*1*
*2*
*4*
*5*
*6*
*7*
**                    ：空文字列です
*1*
*2*
*4*
```



```
*5*
*6*
*7*
**                    :空文字列です
*1*
*2*
*4*
*5*
*6*
*7*
**                    :空文字列です
 :
 :
```
(終わりません)

このようになったときは強制中断するしか方法はありません。強制中断の方法を覚えていますか。CTRL キーを押しながらCキーを打ってください。

<INRPT> 割込み要求が発生しました

というエラーが出て，処理が中断するはずです。

困ったときの「CTRL/C」頼み！

- FOR命令は最終値を指定しないことによって無限ループを作ることができる
- FOR命令の無限ループにはQUIT命令を入れておかないと終了しない
- 無限ループは，CTRL/Cで強制中断できる

7,名前のリスト

登録番号だけではなく，名前も同時に出力できるようにしましょう。

list 8-6

```
MB036   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1985.5.17;$ORDER ｶﾝｽｳ ﾉ ﾚｲ, FOR ﾒｲﾚｲ ﾆ ﾖﾙ ｸﾘｶｴｼ
        ;
        S NO=""
        F QQ=0:0 S NO=$O(^MBM(NO)) W !,NO,?5,^MBM(NO,0) I NO="" Q
        Q
```

このルーチンではグローバル変数 ^MBMの添え字をXではなく，NO にしましたが，ルーチンの意味はまったく同じです。実行します。

```
>D MB036↵

1    ｲﾜﾂｷ ﾋﾛﾐ
2    ﾔﾏﾀﾞ ﾀﾛｳ
4    ﾋﾗﾉ ｼﾞｭﾝｺ
5    ｷﾀﾑﾗ ｹﾝ
6    ﾊｼﾓﾄ ｱﾂｺ
7    ﾋﾗﾀ ﾐﾈｺ

<UNDEF> 定義されていない変数が参照されました
 1) +4^MB036:: F QQ=0:0 S NO=$O(^MBM(NO)) W !,NO,?5,^MBM(NO,0) I NO="" Q
 0) *XECUTE*^MB036:: D MB036
<UNDEF>+4^MB036:::4:1 F QQ=0:0 S NO=$O(^MBM(NO)) W !,NO,?5,^MBM(NO,0) ...
>
```

オッと！ 〈UNDEF〉 エラーが出てしまいました。なぜでしょうか？ 変数 NO の値を調べましょう。

```
>W "*",NO,"*"↵
**
>
```

NO には空文字列が入っています。もうひとつ確かめてください。

```
>W ^MBM(NO,0)↵

<UNDEF> 定義されていない変数が参照されました
 0) *XECUTE*^MB036:: W ^MBM(NO,0)
<UNDEF> *XECUTE*^MB036:::4:1 W ^MBM(NO,0)
>
```

分かりましたか？

変数 NO="" のときに ^MBM(NO,0) (=^MBM("",0))を表示しようとしました。このような変数は存在しないのでエラーになったのです。NO が空文字列のときは表示をせずに QUIT させなくてはなりません。

list 8-7

```
MB037   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1985.5.17;$ORDER ｶﾝｽｳ ﾉ ﾚｲ, FOR ﾒｲﾚｲ ｺ ﾖﾙ ｸﾘｶｴｼ
        ;
        S NO=""
        F QQ=0:0 S NO=$O(^MBM(NO)) I NO="" Q ↑W !,NO,?5,^MBM(NO,0)
        Q
```

WRITE命令の前にIFとQUIT命令を置きました

ここには空白が2つ必要です

IF 命令と QUIT 命令を WRITE 命令の前に移動しました。

QUIT 命令のように引数をもたない命令の後に，他の命令が続くときには，QUIT とその命令の間に空白をふたつ以上挿入しなくてはならないので注意してください。

実行してみましょう。

```
>D MB037↵

>
```

何も表示されず，エラーにもなりません。何も表示されないのは，IF 命令，QUIT 命令，WRITE 命令の並んでいる順序に原因があります。IF 命令の NO="" という条件が正しい（「真」）のときと，誤り（「偽」）のときで分けて考えましょう。

①NO="" が「偽」のとき

IF 命令はそれより右にある命令の実行をすべて禁止してしまいます。したがって QUIT 命令も WRITE 命令も実行されません。

```
F QQ=0:0 S NO=$O(^MBM(NO)) I NO="" Q  W !,NO,?5,^MBM(NO,0)
                                   ↑
                          この２つの命令は実行されません
```

②NO="" が「真」のとき

IF 命令は右にある命令の実行を許可します。そこでまず QUIT 命令を実行します。すると，これは左端にある FOR 命令を終了させます。その結果これより右にある WRITE 命令は実行されません。

```
F QQ=0:0 S NO=$O(^MBM(NO)) I NO="" Q  W !,NO,?5,^MBM(NO,0)
                                   ↑
                          このWRITE命令は実行されません
```

結局，WRITE 命令は永久に実行されないことになります。私たちに必要なのは

NO="" が「偽」のときに QUIT 命令は実行せずに WRITE 命令は実行する

NO="" が「真」のときに QUIT 命令を実行して WRITE 命令は実行しない

ということです。これには IF 命令でなく「後付け条件」という方法を用います。

「後付け条件」は，命令語（SET 命令のSや，QUIT 命令のQ）の直後にコロン『:』を付け，さらに条件を付けるという構文です。たとえば

```
I NO="" Q
I $D(^X(NO))=1 S A=^X(NO)
```

という IF 命令による条件付けは，それぞれ

```
Q:NO=""
S:$D(^X(NO))=1 A=^X(NO)
```

とすることができます。

「後付け条件」と IF 命令の違いはつぎのとおりです。

①IF 命令は，その命令より右にあるすべての命令を条件付ける

②後付け条件は，条件の付いている命令だけを条件付け，その右にある命令には直接には影響しない

したがって

```
Q:NO=""  W !,NO,?5,^MBM(NO,0)
```

とすると，NO=""が「偽」のときは QUIT 命令は実行されませんが，WRITE 命令は実行され，NO=""が「真」のとき QUIT 命令が実行されて FOR 命令が終了するので WRITE 命令は実行されないことになります。

後付け条件が付いていても，QUIT 命令と WRITE 命令の間には空白が2個必要です。

この空白をしばしば忘れるので注意してください。

list 8-8

```
MB038   ;YS,ｼｮｸｲﾝ･ﾒｲｾﾎﾞ;1985.5.17;$ORDER ｶﾝｽｳ ﾉ ﾚｲ, FOR ﾒｲﾚｲﾎﾞ ｺ ﾖﾙ ｸﾘｶｴｼ
        ;
        S NO=""
        F QQ=0:0 S NO=$O(^MBM(NO)) Q:NO=""  W !,NO,?5,^MBM(NO,0)
        Q
```

後付け条件です

ここは空白２個です

というルーチンができあがります。

実行してみましょう。

```
>D MB038↵

1     ｲﾜｻｷ ﾋﾛﾐ
2     ﾔﾏﾀﾞ ﾀﾛｳ
4     ﾋﾗﾉ ｼﾞｭﾝｺ
5     ｷﾀﾑﾗ ｹﾝ
6     ﾊｼﾓﾄ ｱﾂｺ
7     ﾋﾗﾀ ﾐﾎｺ
>
```

エラーも出ずうまく行ったようです。他のデータも表示できるようにしましょう。

list 8-9

```
MB039   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1985.3.21;ﾃﾞｰﾀ ﾉ ﾘｽﾄｱｯﾌﾟ  ($ORDER ｶﾝｽｳ)
        ;
HAJIME  S NO=""
        F QQ=0:0 S NO=$O(^MBM(NO)) Q:NO="" W !,NO,?5,^MBM(NO,0),?20,^MBM(NO,1),
    -->?25,^MBM(NO,2),?30,^MBM(NO,3)
        W !,"*** ｵﾜﾘ ***"
        Q
```

空白２個忘れないこと

このルーチンは FOR 命令の無限ループと $ORDER を用いてグローバル変数を検索する基本的で典型的なパターンなので確実に頭に入れてください。

```
>D MB039↵

1     ｲﾜｻｷ ﾋﾛﾐ          F     26    ｺｳﾎｳｼｼﾂ
2     ﾔﾏﾀﾞ ﾀﾛｳ          M     19    ﾍﾝｼｭｳﾌﾞ
4     ﾋﾗﾉ ｼﾞｭﾝｺ         F     27    ﾋｼｮｶ
5     ｷﾀﾑﾗ ｹﾝ           M     36    ｾｯｹｲﾌﾞ
6     ﾊｼﾓﾄ ｱﾂｺ          F     22    ﾀﾞｲ2 ｴｲｷﾞｮｳﾌﾞ
7     ﾋﾗﾀ ﾐﾎｺ           F     22    ｹｲﾘﾌﾞ
*** ｵﾜﾘ ***
>
```

- 命令には後付け条件を付けることができる
- 後付け条件はその命令にのみ有効で，その命令の右側にある命令の実行には影響しない
- 引数のない命令と次の命令の間には２つ以上の空白を入れる
- 変数の添え字を検索する基本的パターン：
  S X="" F QQ=0:0 S X=$O(A(X) Q:X="" .....

8,DO命令の利用

$ORDER 関数を用いて，いろいろな条件でデータをリストするルーチンを書いてみましょう。

①女性のみのリスト

list 8-10

```
MB040   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1985.3.22;ｼﾞｮｾｲ ﾉﾐ ﾉ ﾃﾞｰﾀ ﾉ ﾘｽﾄｱｯﾌﾟ($ORDER ｶﾝｽｳ)
        ;
HAJIME  S NO=""
        F QQ=0:0 S NO=$O(^MBM(NO)) Q:NO=""  I ^MBM(NO,1)="F" W !,NO,?5,^MBM(NO,0
     -->),?20,^MBM(NO,1),?25,^MBM(NO,2),?30,^MBM(NO,3)
        W !,"*** ｵﾜﾘ ***"
        Q
```

このルーチンは IF 命令でデータを表示するか否かの条件付けをしていること以外は「全データリスト」のルーチンと同じものです。

②男性のみのリスト

list 8-11

```
MB041   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1985.3.22;ﾀﾞﾝｾｲ ﾉﾐ ﾉ ﾃﾞｰﾀ ﾉ ﾘｽﾄｱｯﾌﾟ ($ORDER ｶﾝｽｳ)
        ;
HAJIME  S NO=""
        F QQ=0:0 S NO=$O(^MBM(NO)) Q:NO=""  I ^MBM(NO,1)="M" D HYOJI
                                                              ①
   ②    W !,"*** ｵﾜﾘ ***"
        Q
HYOJI   W !,NO,?5,^MBM(NO,0),?20,^MBM(NO,1),?25,^MBM(NO,2),?30,^MBM(NO,3)
        Q
```

このルーチンは基本的には「女性のみのリストのルーチン」と同じですがデータの表示を HYOJI という別の行で行っています。

①の D HYOJI を実行すると，行 HYOJI へ飛びます。そして行 HYOJI＋1 に QUIT 命令があるので，そこまで行くとまた①の D HYOJI のところへ戻ります。

D HYOJI の右にはもう命令がないので，この行は終わりですが，FOR 命令が左端にあるので，また SET 命令から繰り返します。

女性のみのリストのルーチン MB040 のように1行ですべて仕事を片付けるか，このように DO 命令を用いて他の行を呼び出してそこで作業するかは作業の量と好みによって決められます。

ルーチンの1行は255文字以内という制限があるので，作業が複雑になれば DO 命令で拡張せざるを得ません。一般に DO 命令を用いた方が普遍性，拡張性が高いと考えられます。

②の QUIT 命令はなぜ必要なのでしょうか？

これは，Chapter 6-9で解説したように，行 HAJIME から HAJIME+3 までと，行 HYOJI 以降では仕事のレベルが違うからです。

この QUIT 命令を取ってしまうと，

list 8-12

```
MB042   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1985.3.22;ﾀﾞﾝｾｲ ﾉﾐ ﾉ ﾃﾞｰﾀ ﾉ ﾘｽﾄｱｯﾌﾟ ($ORDER ｶﾝｽｳ)
        ;
HAJIME  S NO=""
QUIT    F QQ=0:0 S NO=$O(^MBM(NO)) Q:NO=""  I ^MBM(NO,1)="M" D HYOJI
がない  W !,"*** ｵﾜﾘ ***"
        →
HYOJI   W !,NO,?5,^MBM(NO,0),?20,^MBM(NO,1),?25,^MBM(NO,2),?30,^MBM(NO,3)
        Q
```

となります。実行してみましょう。

```
>D ^MB042↵

2     ﾔﾏﾀﾞ ﾀﾛｳ          M     19    052-555-6666
5     ｷﾀﾑﾗ ｹﾝ           M     36    06-999-8888
*** ｵﾜﾘ ***

<UNDEF> 定義されていない変数が参照されました
 1) +6^MB042:: W !,NO,?5,^MBM(NO,0),?20,^MBM(NO,1),?25,^MBM(NO,2),?30,...
 0) *XECUTE*^MB042:: D ^MB042
<UNDEF>+6^MB042:::4:1 W !,NO,?5,^MBM(NO,0),?20,^MBM(NO,1),?25,^MBM(NO,...
>
```

このようにエラーになってしまいます。

データをすべて調べ終わり，$ORDER 関数の結果 NO が空文字列となると QUIT 命令が実行され，FOR 命令が終了します。そして，次の行 HAJIME+2 の WRITE 命令を実行して ' ***オワリ*** ' と表示します。

ここまでは期待通りでした。しかし，まだその後に行が2行残っているのでそこで終わらず，再度行 HYOJI を実行してしまいます。ところが，変数 ^M

BM(NO,0)の添え字の NO が空文字列のままなのでエラーになってしまったのです。

結局，'***オワリ***'と表示した後はこのルーチンの仕事は終了したわけですから，ここでルーチン全体が終わるという意味の QUIT 命令を入れておく必要があるのです。

行 HAJIME から行 HAJIME+2 までは，次々に登録番号を調べて行 HYOJI を呼び出す仕事をしています。一方，行 HYOJI 以降は，渡された登録番号 NO の人のデータを表示しています。

このように，このルーチンはふたつのレベルの違う仕事から構成されています。Chapter 6-9 で解説したように，レベルの違う仕事をする部分の間には QUIT 命令を入れることを忘れないでください。

他のルーチンのリストもあげておきます。今後は一般性を重んじて DO 命令を使用します。

③指定された年齢未満のデータのリスト

list 8-13

```
MB043   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1985.3.22;ｱﾙ ﾈﾝﾚｲ ﾐﾏﾝ ﾉ ﾋﾄ ﾉ ﾃﾞｰﾀ ﾉ ﾘｽﾄｱｯﾌﾟ
        ;
HAJIME  R !,"ﾈﾝﾚｲ ﾊ ? ",AGE
        S NO=""
        F QQ=0:0 S NO=$O(^MBM(NO)) Q:NO=""  I ^MBM(NO,2)<AGE D HYOJI
        W !,"*** ｵﾜﾘ ***"
        Q
HYOJI   W !,NO,?5,^MBM(NO,0),?20,^MBM(NO,1),?25,^MBM(NO,2),?30,^MBM(NO,3)
        Q
```

④名前の一部による検索

list 8-14

```
MB044   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1985.3.22;ﾅﾏｴ ｺ ﾖﾙ ｹﾝｻｸ
        ;
HAJIME  R !,"ﾅﾏｴ ﾊ ? ",NAME
        S NO=""
        F QQ=0:0 S NO=$O(^MBM(NO)) Q:NO=""  I ^MBM(NO,0)[NAME D HYOJI
        W !,"*** ｵﾜﾘ ***"
        Q
HYOJI   W !,NO,?5,^MBM(NO,0),?20,^MBM(NO,1),?25,^MBM(NO,2),?30,^MBM(NO,3)
        Q
```

以上のルーチンは，FOR 命令，QUIT 命令，DO 命令，$ORDER 関数の組み合わせがすべて同一形式になっています。これが基本パターンなので，十分理解してください。

- ルーチンの中で DO 命令を使うことにより，ルーチンの一部を呼び出すことができる
- DO命令は指定された行からルーチンの終わり，またはQUIT命令までを実行し，またもとのDO命令へ戻る
- FOR 命令と DO 命令の組み合わせで，ルーチンの一部を繰り返して実行することができる
- QUIT 命令がルーチンの途中にあると，そのルーチンはそのQUIT命令のところで終了する
- QUIT 命令がないと，ルーチンを余分に実行してしまうことがある
- ルーチンの１行の長さは255文字以内

9,平均年齢の計算

平均年齢を求めるルーチンも $ORDER を用いる形に変更しましょう。

list 8-15

```
MB045   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1986.2.13;ﾍｲｷﾝ ﾈﾝﾚｲ ﾉ ｹｲｻﾝ
        ;
HAJIME  S GOKEI=0
        S NO=""
        F J=1:1 S NO=$O(^MBM(NO)) Q:NO=""  S GOKEI=GOKEI+^MBM(NO,2)
        S NINZU=J-1
        W !,"ﾍｲｷﾝ ﾈﾝﾚｲ = ",GOKEI/NINZU
        Q
```

Chapter 6-7 のルーチン MB023 と比較してください。

GOKEI を０にセットして，データごとに S GOKEI=GOKEI+ ^MBM(NO,2)という計算をするのは同じことです。

一方，登録番号 NO を空文字列にセットして，$ORDER 関数で次の NO を取り出すことは前節８のリストのルーチンと同じです。

ひとつ異なるのは，FOR 命令が

```
F J=1:1
```

となっていて，この変数Jの値を用いて人数 NINZU を算出していることです。

```
F QQ=0:0
```

という形の FOR 命令は，繰り返す回数に意味がありませんでした。しかし，このルーチンの場合，変数Jの値は繰り返すごとに1ずつ増すので，繰り返した回数を数えています。

ただし最後に登録番号 NO が空文字列になるときもその回数に含まれます。このため， FOR 命令の繰り返しが終了した時点で， Jの値は**登録されている人数**＋1になっています。そこで，FOR 命令の次の行で，

```
S NINZU=J-1
```

として，人数を計算しています。実行しましょう。

```
>D ^MB045↵

ﾍｲｷﾝ ﾈﾝﾚｲ = 25.333333333333
>
```

最後に1を引かずに人数を求める方法を紹介します。繰り返しのカウントを，1からではなく0から始めます。すると変数の値はちょうど登録人数に一致します。

```
F NINZU=0:1
```

次のルーチンを見てください。

list 8-16

```
MB046   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1986.2.13;ﾍｲｷﾝ ﾈﾝﾚｲ ﾉ ｹｲｻﾝ
        ;
HAJIME  S GOKEI=0          初期値 0
        S NO=""
        F NINZU=0:1 S NO=$O(^MBM(NO)) Q:NO=""  S GOKEI=GOKEI+^MBM(NO,2)
        W !,"ﾍｲｷﾝ ﾈﾝﾚｲ = ",GOKEI/NINZU
        Q
```

- 無限に繰り返すFOR命令でも，繰り返した回数が意味をもつときには

  F J＝1:1　あるいは　F K＝0:1

  という形を用いる

Chapter 9

データを並べ替える

この章ではまず，変数の添え字に文字列データを用いることを解説します。

次に「索引グローバル」を，さらに変数に複数の添え字がある場合の $ORDER 関数の働きを解説します。

最後にこれらを利用して50音順，年齢順などで職員名簿を並べ替えて表示する方法を示します。

1,文字列添え字

今まで，変数の添え字はすべて数値でした。しかしグローバル変数にもローカル変数にも，添え字として数字以外の文字を用いることができます。[2)]

次の例を試してください。

```
>K ^PET↵                                  ：変数^PET とその子供達を削除します
>S ^PET("OWL")="ﾌｸﾛｳ"↵                    ：改めて文字列添え字付き変数^PETを
>S ^PET("PARROT")="ｵｳﾑ"↵                    セットします
>S ^PET("SEAGULL")="ｼﾞｮﾅｻﾝ"
>S ^PET("PEACOCK")="ｸｼﾞｬｸ"↵
>S ^PET("ﾗｸﾀﾞ")="CAMEL"↵                   ：カナの添え字
>S ^PET("ｷﾘﾝ")="GIRAFFE"↵
>S ^PET(3)="ﾌﾞｰ ﾌｰ ｳｰ"↵                    ：これは数字の添え字
```

セットした変数の値を確かめます。

```
>W ^PET("PEACOCK")↵          ：WRITE命令
ｸｼﾞｬｸ                        引用符『"』を忘れないこと
>W ^PET("ﾗｸﾀﾞ")↵
CAMEL
>W ^PET(3)↵
ﾌﾞｰ ﾌｰ ｳｰ
>
```

このように数字以外の文字を，数値の添え字とまったく同じように用いることができます。

ただし添え字として直接，文字の値を書くときには『"』(引用符) で囲まなくてはなりません。そうしないと MUMPS は文字の値を変数の名前として扱います。

```
>W ^PET(OWL)↵

<UNDEF> 定義されていない変数が参照されました
 0) *XECUTE*:: W ^PET(OWL)
<UNDEF>*XECUTE*:::4:0 W ^PET(OWL)
>
```

変数 OWL が定義されていないので 〈UNDEF〉 エラーになりました。

```
>S OWL="ﾗｸﾀﾞ"↵              ：変数OWLに'ラクダ'という値をセットします

>W ^PET(OWL)↵               ：これはW ^PET("ラクダ")と等価です
CAMEL
>
```

- 添え字には数値だけではなく，文字列データも用いることができる
- 文字列を添え字として直接タイプするときには，引用符で囲むさもないと，それは変数の名前として解釈される

■notes

2) 標準MUMPSでは，ASCII文字（アルファベットや記号）を添え字にできます。日本語MUMPS標準化委員会による仕様では漢字（を含む全角文字）やカナも変数の添え字として用いることができます。SP-MUMPS でも漢字やカナを用いることができます

2,名前による索引グローバル

ではこの文字列添え字を用いて次のようなグローバル変数を作りましょう。

```
^MBIDX1(名前)=登録番号        →    ^MBIDX1("ｲﾜｻｷ ﾋﾛﾐ")=1
                                   ^MBIDX1("ﾔﾏﾀﾞ ﾀﾛｳ")=2
                                   ^MBIDX1("ﾋﾗﾉ ｼﾞｭﾝｺ")=4
                                         :
                                         :
```

^MBIDX1というのは，職員名簿システムのグローバル変数で，索引(index)の1番という意味で命名しました

このグローバル変数は今後とても重要な意味をもつものです。その作り方や使い方を理解するために，人間の手作業によるデータ処理の例を考えましょう。

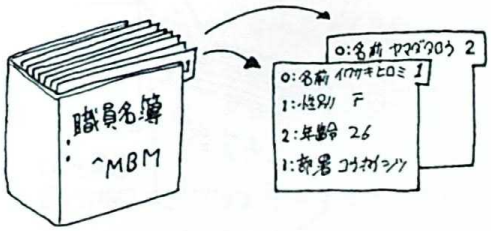
今，職員名簿の MBM というファイルがあるものとします。このファイルは，たくさんの職員登録カードからなり，1枚1枚のカードはひとり1人の職員に対応しています。

各カードにはそれぞれ職員の番号が見出しとして付いています。N番の職員登録カードを MBM(N) と呼ぶことにします。

カードはいくつかの欄に分かれていて，その各欄には0から3までの番号が付いています。すべてのカードの第0欄には職員の名前，第1欄には性別，第2欄には年齢，そして第3欄には所属部署が記入されています。

fig 9-1　ファイルMBM

fig 9-2　カードME



さて，このファイルにはカードがたくさんあります。このカードの中からある名前の人のカードを取り出そうと考えます。

もっとも単純な方法は，1枚1枚カードをめくって，各カードに書いてある職員の名前を調べることです。これは前章で FOR 命令と $ORDER 関数の組み合わせで用いた方法です。

しかしこれはカードの枚数が少ない場合はよいのですが，枚数が多くなるといちいち時間がかかってたまりません。

そこで登場するのが「索引」です。

図書館の文献索引カードのように，名前を見出しにしたカードを作り登録番号を書いておけば，その番号から必要な人のデータを調べることができます。

前述したグローバル変数 ^MBIDX1 は，ちょうどこの索引カードの役割をすることができます。たとえばある名前の人の年齢を知りたければ

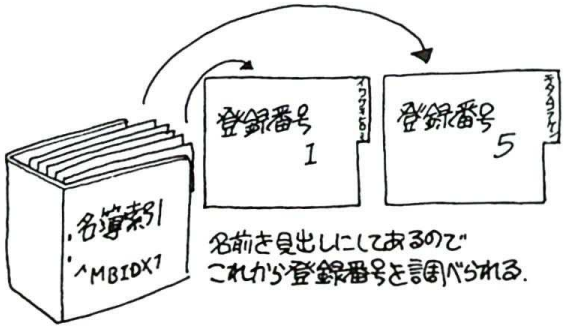
```
R !,"ナマエ ? ",NAMAE
S NO=^MBIDX1(NAMAE)
W !,"ネンレイ = ",^MBM(NO,2)
```

とすればよいでしょう。

索引カード ^MBIDX1を作るには，1回は職員名簿 ^MBMに登録されているすべての人の名前を調べなくてはなりません。この作業は前章のリストのプログラムをそのまま流用できます。

```
S NO=""
F QQ=0:0 S NO=$O(^MBM(NO)) Q:NO=""  W !,NO,?5,^MBM(NO,0),...(省略)
```

fig 9-3　索引カード



『W！…』以下の部分でデータを表示する代わりに，登録番号 NO を用いて名前を取り出し

```
S NAMAE=^MBM(NO,0)
```

そして，名前と登録番号から索引をセットします。

```
S ^MBIDX1(NAMAE)=NO
```

このふたつの SET 命令をひとつにまとめて

```
S NAMAE=^MBM(NO,0),^MBIDX1(NAMAE)=NO
```

とすることができます。これは引数をふたつもった SET 命令です。

次にグローバル変数 ^ MBIDX1を作るルーチンのリストを掲げます。

list 9-1

```
MB047   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1985.3.22;ﾅﾏｴ ﾆ ﾖﾙ ｻｸｲﾝ ﾌｧｲﾙ ^MBIDX1 ｦ ﾂｸﾙ
        ;
HAJIME  K ^MBIDX1 S NO=""
        F QQ=0:0 S NO=$O(^MBM(NO)) Q:NO=""  S NAMAE=^MBM(NO,0),^MBIDX1(NAMAE)=NO
        Q             ⓘ                ⓘⓘ             ⓘⓘⓘ                  ⓘⓥ
```

このルーチンの働きをもう一度解説します。

① 最初に古い索引ファイル ^ MBIDX1を削除します

② $ORDER 関数で，最初の登録番号を得るために，登録番号を空文字列にします

③ FOR 命令でⓘからⓘⓥまでの処理を繰り返します。

ⓘ 名簿ファイル ^ MBMに登録されている人の登録カードの番号 NO を調べます

ⓘⓘ もしカードがない（登録番号が空文字列）ならばこの繰り返しは終了します

ⓘⓘⓘ 登録番号 NO の人の名前を取り出します（変数 NAMAE にセットします）

ⓘⓥ 名前 NAMAE を見出し（添え字）にして，カード（グローバル変数 ^ MBIDX1)を作ります（セットします）

ところで，行 HAJIME では，以前に作った索引グローバル ^ MBIDX1を削除（KILL）していることに注意してください。

ここで KILL しておかないと，もし以前に作った索引があると，それと新しい索引が混ざってしまい，利用するときに矛盾を生じます。

古い索引カードを用いるのは，古い電話番号簿を使って間違い電話をかけるようなものです。

なお存在しない変数を KILL するのはエラーになりません。MUMPS はその KILL 命令を無視して何もしません。

ルーチンを試しましょう。

```
>D ^MB047↵                    ：グローバル^MBIDX1を作ります
>                             ：このルーチンは何も表示しません
```

さて「グローバル　リスト　ユーティリティ（%GL)」で，いま作成したグローバル変数 ^MBIDX1の内容を調べます。

```
>D ^%GL↵                      ：グローバル リスト ユーティリティを起動します

グローバル　リスト　ユーティリティ

グローバルの選択: ^MBIDX1↵   ：リストしたい変数^MBIDX1を指定
出力装置 <0> ↵
スクロールさせますか? <N> ↵
^MBIDX1("ｲﾜｻｷ ﾋﾛﾐ")="1"
^MBIDX1("ｷﾀﾑﾗ ｹﾝ")="5"
^MBIDX1("ﾊｼﾓﾄ ｱﾂｺ")="6"
^MBIDX1("ﾋﾗﾀ ﾐﾎｺ")="7"
^MBIDX1("ﾋﾗﾉ ｼﾞｭﾝｺ")="4"
^MBIDX1("ﾔﾏﾀﾞ ﾀﾛｳ")="2"

グローバルの選択: ^ ↵        ：何も入力せずに↵キーのみを打つと
>                             ルーチンが終了します
```

期待通りにセットされていますね。

- 名前を添え字にしたグローバル変数は，名前による索引ファイルとして使用できる
- 存在しない変数をKILLしてもエラーにならない。MUMPSはそのKILL命令を無視する

3,索引グローバルを利用した検索プログラム

次に示すのは，索引ファイル ^MBIDX1を用いて名前による検索をするルーチンです。データをすべて調べる方法では登録人数が数百人にもなると，検索時間が問題になります。

索引ファイルを用いる方法では，何万人に達してもほとんど待つことなく検索できます。

```
>W ^MBIDX1("ｷﾀﾑﾗ ｹﾝ")↵
5                                  :'キタムラケン'さんの登録番号が5とわかります
>W ^MBM(5,0)↵
ｷﾀﾑﾗ ｹﾝ                            :登録番号5の人の名前は'キタムラケン'です
>
```

この操作をルーチンにしておきましょう。

list 9-2

```
MB048   ;YS､ｼｮｸｲﾝ･ﾒｲﾎﾞ;1986.2.1;ｻｸｲﾝ ﾌｧｲﾙ ﾖ ﾘﾖｳｼﾀ ﾅﾏｴ ﾆﾖﾙ ｹﾝｻｸ
        ;
B       R !,"ﾅﾏｴ ? ",NAMAE
        I $D(^MBIDX(NAMAE))=0 W "   ﾐﾂｶﾘﾏｾﾝ" Q
        S NO=^MBIDX(NAMAE)
        W !,NO,?5,^MBM(NO,0),?20,^MBM(NO,1),?25,^MBM(NO,2),?30,^MBM(NO,3)
        Q
```

I $D(^MBIDX(NAMAE))=0 W"の行では，入力した名前の人がいるかどうかを調べています。もし登録されていなければ，'ミツカリマセン'と表示してこのルーチンは終了します。

このルーチンは1回検索すると終了してしまいます。繰り返して検索できるように改造したのがリスト9-3のルーチンです。

名前を入力せずに，『＊』記号を入力すると行Aの QUIT 命令が実行され，FOR 命令が終了します。指定した名前が存在しなくても，変数 NAMAE には『＊』記号でないデータが入っているので，その場合には FOR 命令は終了しません。

list 9-3

```
MB049   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1986.2.1;ｻｸｲﾝ ﾌｧｲﾙ ﾖ ﾘﾖｳｼﾀ ﾅﾏｴ ﾆﾖﾙ ｹﾝｻｸ
        ;
A       F QQ=0:0 D B I NAMAE="*" Q
        Q
B       R !,"ﾅﾏｴ ? ",NAMAE
        I NAMAE="*" Q
        I $D(^MBIDX(NAMAE))=0 W "   ﾐﾂｶﾘﾏｾﾝ" Q
        S NO=^MBIDX(NAMAE)
        W !,NO,?5,^MBM(NO,0),?20,^MBM(NO,1),?25,^MBM(NO,2),?30,^MBM(NO,3)
        Q
```

4,文字列添え字の並ぶ順序

次に文字列添え字を $ORDER 関数で捜したときの順序について説明します。さきほどセットしたグローバル変数 ^PETを利用して考えましょう。

```
>W $O(^PET(""))↵
3
>W $O(^PET(3))↵          ：最初の添え字です
OWL
>W $O(^PET("OWL"))↵
PARROT
>
```

FOR 命令を使ってみましょう。

```
>S X="" F QQ=0:0 S X=$O(^PET(X)) Q:X=""  W !,X↵

3
OWL
PARROT
PEACOCK
SEAGULL
ｷﾘﾝ
ﾗｸﾀﾞ
>
```

いつものパターンです
覚えてください

最初が数字です。次にアルファベット，そしてカタカナはそのあとです。数字は今まで見てきたように数字の小さい順です。

アルファベットやカタカナは「辞書配列」の順序になります。たとえば 'A

B'は'AAA'のあとになります。

ただし，小文字のaは大文字のZのあとになり，カタカナではヲが最初で次に小文字のァィゥェォャュョッが現われそれから普通のアイウエオが続くというように普通の辞書とは若干異なった配列になります。

これらの順序は，ひとえにASCII (American Standard Code for Information Interchange) コードとかJISコードといわれる規格で決まった配列によるものです。規格に従う以上我慢しなくてはならないことです。

このコード表については file 6 でまとめて触れます。(かな・漢字を用いる場合もJIS漢字コードという規格があり，それに従った順序になります)[3]。

普通はそれほど厳密な辞書配列を必要とはしないのでこれで実用上十分でしょう。

文字列を添え字に使う場合には，文字列の長さは63文字という制限があるので少し注意が必要です。[4]

- 文字列添え字を$ORDER関数で検索すると「辞書配列」の順になる
- アルファベット，数字，カタカナの文字列添え字を$ORDER関数で検索すると，数字，アルファベット，カタカナの順になる

5, 名前順のリスト

$ORDER 関数を利用して，さきほど作成した索引グローバル ^MBIDX1 の添え字を検索すれば，名前順に登録番号を調べることができそうです。

```
>W $O(^MBIDX1(""))↵      ：名前順で，最初の人の名前を捜します
ｲﾜｻｷ ﾋﾛﾐ                 ：最初の人の名前は'イワサキヒロミ'さんでした
```

■notes

3) 日本語MUMPS標準化委員会によると，アルファベット，カタカナ，漢字などが混在しているときは，$ORDER関数で並べた場合の順序は決っていません。SP-MUMPSでは，漢字はアルファベットとカタカナの間に入ります

4) 標準MUMPSではひとつの変数の変数名と，文字列添え字の長さの合計が63文字以内という制限があります。SP-MUMPSでは合計が255文字（漢字は1文字を2として計算）までです。

```
>W ^MBIDX1("ｲﾜｻｷ ﾋﾛﾐ")↵          ：その人の登録番号を調べます
1                                ：登録番号は１でした
>W $O(^MBIDX1("ｲﾜｻｷ ﾋﾛﾐ"))↵      ：'イワサキヒロミ'さんの次の名前を捜します
ｷﾀﾑﾗ ｹﾝ                          ：'キタムラケン'さんでした
>W ^MBIDX1("ｷﾀﾑﾗ ｹﾝ")↵           ：この人の登録番号は，
5                                ：５番でした
>
```

何をしたのかよく考えてください。この作業を繰り返せば全員の名前と登録番号を名前順にリストできます。

たとえば次のようなルーチンが考えられます。

list 9-4

```
MB050   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1986.1.31;ｻｸｲﾝ ﾌｧｲﾙ ﾘﾖｳ ﾉ ﾚｲ
        ;
HAJIME  S NAMAE=""
        F QQ=0:0 S NAMAE=$O(^MBIDX1(NAMAE)) Q:NAMAE=""  W !,NAMAE,?20,^MBIDX1(NA
      -->MAE)
        Q
                          ⓘ                   ⓘⓘ           ⓘⓘⓘ
```

このルーチンの働きは次のとおりです。

①$ORDER で最初の名前を捜すため，NAMAE に空文字列をセットします

②以下の作業を繰り返します

ⓘ グローバル変数 ^ MBIDX1の添え字をもとに名前を調べます

ⓘⓘ 名前がなければここで繰り返しは終わりです

ⓘⓘⓘ 名前があれば，その名前と ^ MBIDX1の値（＝登録番号）を表示します

実行します。

```
>D ^MB050↵

ｲﾜｻｷ ﾋﾛﾐ        1
ｷﾀﾑﾗ ｹﾝ         5
ﾊｼﾓﾄ ｱﾂｺ        6
ﾋﾗﾀ ﾐﾎｺ         7
ﾋﾗﾉ ｼﾞｭﾝｺ       4
ﾔﾏﾀﾞ ﾀﾛｳ        2←[登録番号です]
>       ↑
   [名前NAMAEです]
```

登録番号がわかれば，その人の全データを表示するのは簡単です。

list 9-5

```
MB051   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1985.3.22;ﾅﾏｴ ｼﾞｭﾝ ﾉ ﾃﾞｰﾀ ﾘｽﾄ
        ;
HAJIME  S NAMAE=""
        F QQ=0:0 S NAMAE=$O(^MBIDX1(NAMAE)) Q:NAMAE=""  S NO=^MBIDX1(NAMAE) D HY
    -->OJI                                                  ⅲ                ⅳ
        Q
HYOJI   W !,NO,?5,^MBM(NO,0),?20,^MBM(NO,1),?25,^MBM(NO,2),?30,^MBM(NO,3)
        Q                          ⅴ
```

このルーチンと前のルーチンの違いは次のとおりです。

ⅲで，データを表示せず，登録番号 NO を取り出します

ⅳでは，データを表示するために行 HYOJI を呼び出します

ⅴで，登録番号 NO の人のデータを表示します

さあ，実行してください。

```
>D ^MB051⏎

1       ｲﾜｻｷ ﾋﾛﾐ        F     26     ｺｳﾎｳｼﾂ
5       ｷﾀﾑﾗ ｹﾝ         M     36     ｾｯｹｲﾌﾞ
6       ﾊｼﾓﾄ ｱﾂｺ        F     22     ﾀﾞｲ2 ｴｲｷﾞｮｳﾌﾞ
7       ﾋﾗﾀ ﾐﾎｺ         F     22     ｹｲﾘﾌﾞ
4       ﾋﾗﾉ ｼﾞｭﾝｺ       F     27     ﾋｼｮｶ
2       ﾔﾏﾀﾞ ﾀﾛｳ        M     19     ﾍﾝｼｭｳﾌﾞ
>
```

確かに名前順にリストされました。

- 名前順にデータ・リストを作りたいときには，名前を添え字にした変数を作りそれを $ORDER 関数で検索する。

6,年齢順に並べ替える

次に年齢順にデータを並べて見ましょう。これには名前の代わりに年齢を添え字とし，登録番号をデータとするグローバル変数を作ればよいことがすぐに理解できるでしょう。ここでは

^MBIDX2(年齢)=登録番号 ← MBシリーズの2番目の索引ということでこの名前にしました

というグローバル変数を作ります。まず，この ^MBIDX2を作るルーチンです。さきほどの ^MBIDX1を作るルーチンMB047（リスト9-1）を少し変更しただけです。

list 9-6

```
MB052   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1985.3.22;ﾈﾝﾚｲ ﾆ ﾖﾙ ｻｸｲﾝ ﾌｧｲﾙ ^MBIDX2 ﾖ ｿｸﾙ
        ;
HAJIME  K ^MBIDX2 S NO=""
        F QQ=0:0 S NO=$O(^MBM(NO)) Q:NO=""  S TOSHI=^MBM(NO,2),^MBIDX2(TOSHI)=NO
        Q
```

次にデータを書き出すルーチンです。これも名前順に表示するルーチンMB051を一部変更しただけです。

list 9-7

```
MB053   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1985.3.22;ﾈﾝﾚｲ ｼﾞｭﾝ ﾉ ﾃﾞｰﾀ ﾘｽﾄ
        ;
HAJIME  S TOSHI=""
        F QQ=0:0 S TOSHI=$O(^MBIDX2(TOSHI)) Q:TOSHI=""  S NO=^MBIDX2(TOSHI) D HY
    -->OJI
        Q
HYOJI   W !,NO,?5,^MBM(NO,0),?20,^MBM(NO,1),?25,^MBM(NO,2),?30,^MBM(NO,3)
        Q
```

実行します。

```
>D ^MB052↵        ：検索ファイル^MBIDX2の作成

>D ^MB053↵        ：年齢順のデータリスト

2     ﾔﾏﾀﾞ ﾀﾛｳ          M     19    ﾍﾝｼｭｳﾌﾞ
7     ﾋﾗﾀ ﾐﾎｺ           F     22    ｹｲﾘﾌﾞ
1     ｲﾜｻｷ ﾋｹﾐ          F     26    ｺｳﾎｳｼｯ
4     ﾋﾗﾉ ｼﾞｭﾝｺ         F     27    ﾋｼｮｶ
5     ｷﾀﾑﾗ ｹﾝ           M     36    ｾｯｹｲﾌﾞ
>
```

ちょっと待ってください。大きな問題があります。表示された人の人数が1人足りません。

登録番号6の'ハシモト　アツコ'という人が年齢順のリストに入っていません。

グローバル変数 ^MBIDX2の状態を調べて見ましょう。

```
>D ^%GL↵

グローバル　リスト　ユーティリティ

グローバルの選択: ^MBIDX2↵          :リストしたい変数^MBIDX2を指定
出力装置 <0> ↵                       :↵キーのみを打ちます
スクロールさせますか? <N> ↵
^MBIDX2("19")="2"
^MBIDX2("22")="7"
^MBIDX2("26")="1"
^MBIDX2("27")="4"
^MBIDX2("36")="5"

グローバルの選択:  ^↵
>
```

グローバル変数^MBIDX2にも6という登録番号は出てきません。

ところで'ハシモト　アツコ'さんは年齢22才でした。年齢22才を表す^MBIDX2("22")には'ヒラタ　ミホコ'さんの登録番号7だけが入っています。この2人は同一年齢でした。同一年齢の人が2人以上みつかると

```
S ^MBIDX2(TOSHI)=NO
```

という命令をそのたびに実行します。すると，前にセットされた人の登録番号が後からセットされた人の番号によって置き換えられてしまうのです。ひとつの変数にはひとつのデータ（この場合は1人分の登録番号）しかセットできません。

これを解決するために，索引グローバル変数の構造を次のように変えます。

```
^MBIDX3(TOSHI,NO)=""
```

この構造なら，同じ年齢の人が何人いても索引が重なることがありません。

ただ，添え字がふたつ以上あるとき，$ORDER 関数がどのように働くかを知らなくてはなりません。

- ひとつの変数には原則としてひとつのデータしか入らない
- ある値を持った変数に新しい値をセットすると，古い値が消えてしまう

7,添え字が複数あるときの $ORDER関数の働き

指定した変数の添え字が複数あるとき，$ORDER 関数はどのように働くのでしょう。これは一言で言えば

$ORDERは兄弟の中でのみ働く

ということです。試してみましょう。DA という変数をセットします。

```
>K DA↵                          :ローカル変数DAとその子供達(DA(2), DA(3,4)など
                                 を削除します
>S DA(2)="ｷｻﾗｷﾞ"↵

>S DA(2,14)="ﾊﾞﾚﾝﾀｲﾝ ﾃﾞｰ"↵

>S DA(2,29)="4 ﾈﾝ ｺﾞ 1 ｶｲ ｼｶ ﾅｲ ﾋ"↵

>S DA(3)="ﾔﾖｲ"↵

>S DA(3,3)="ﾐﾐ ﾉ ﾋ"↵

>S DA(3,14)="ﾎﾜｲﾄ ﾃﾞｰ"↵

>S DA(3,21)="ﾀﾌﾞﾝ ｼｭﾝﾌﾞﾝ ﾉ ﾋ"↵

>S DA(4,1)="4 ｶﾞﾂ ﾊﾞｶ!"↵

>S DA(4,8)="ﾊﾅ ﾏﾂﾘ"↵

>S DA(4,15)="ﾀｶﾔﾏ ﾏﾂﾘ"↵

>S DA(4,19)="ﾌﾙｶﾜ ﾏﾂﾘ"↵

>S DA(8,6)="ﾋﾛｼﾏ ｹﾞﾝﾊﾞｸ ｷﾈﾝﾋﾞ"↵

>S DA(8,9)="ﾅｶﾞｻｷ ｹﾞﾝﾊﾞｸ ｷﾈﾝﾋﾞ"↵

>S DA(8,15)="ﾊｲｾﾝ ｷﾈﾝﾋﾞ"↵
```

このようにセットすると，MUMPS は fig9-4 のような変数 DA の「家族」を作ります。その中で，DA(4) や DA(8) は値のない「仮想的な親」であることに注意してください。

添え字がひとつの変数について $ORDER の働きは前章（Chapter 8-3 参照）で解説しましたが，復習しましょう。

```
>W $O(DA(""))↵        ：もっとも小さい添え字の値を調べます
2                     ：もっとも小さい値は 2 です
>W $O(DA(2))↵
3                     ：DA(2)のすぐ上の兄弟はDA(3)なので 3 です
>
```

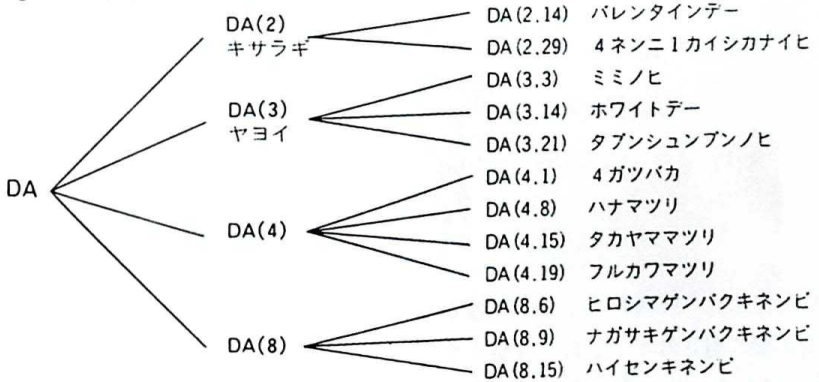
DA(2) の下には DA(2,14) と DA(2,29) という「子供」がいますが，これは「子供」であって「兄弟」ではありません。DA(2) にとってすぐ上の兄弟は DA(3) なので，$O(DA(2)) の結果は 3 になります。

```
>W $O(DA(3))↵         ：DA(3)のすぐ上の兄弟はDA(4)なので 4 です
4
>W $O(DA(4))↵
8
>
```

DA(4) と DA(8) は，それ自身は値を持たないが「子供」があるという「仮想的な親」でした。$D(DA(4)) と $D(DA(8)) はどちらも10になります。

「仮想的」で値がなくても MUMPS の変数としては存在する（$DATA の

fig 9-4　変数DAの家系図



値が0でない）ということです。したがって，DA(3) のすぐ上の「兄弟」は DA(4) なので，$O(DA(3)) の値は4になります。同様に $O(DA(4)) の値は8になります。

そして最大の添え字8を指定すると，空文字列が返されます。

```
>W "*",$O(DA(8)),"*"↵
**
>
```

さて，次は複数の添え字をもつ変数です。

DA(2,14) や DA(2,29) は DA(2) の「子供」
DA(3,14) や DA(3,21) は DA(3) の「子供」

と考えてください。すると

DA(2), DA(3), DA(4) などは「親」同士の「兄弟」
DA(2,14) と DA(2,29) は「子供」同士の「兄弟」
DA(3,14) と DA(3,21) も「子供」同士の「兄弟」

ということになります。そして

DA(2,29) と DA(3,14) は「親」が異なるので「いとこ」
DA(2,29) にとって DA(4) や DA(8) は「伯父」とか「叔母」

となります。

$ORDER 関数はこれらの内で，「兄弟」の中でのみ働きます。したがって

```
>W $O(DA(4,""))↵
1
>W $O(DA(4,1))↵          :「DA(4),」の兄弟の中では最小の添え字
8
>W $O(DA(4,8))↵
15
>W $O(DA(4,15))↵
19
>W $O(DA(4,19))↵         :`DA(4,19)は「DA(4),」の兄弟の中では最大の添え字
                           を持つ変数
>
```

空文字列

となります。'DA(4,' までが共通している変数について，ふたつ目の添え字の値が順に返されているのが分かります。

$O(DA(4, 19)) とすると 'DA(4,' が共通している変数のうちこれが最大の

添え字の変数なので空文字列が返されています。DA(4,19)から見ると，DA(8)は「伯父」DA(8,6)は「いとこ」に当たるので，8や6は返されません。

厳密にいうと次のとおりです。

①$ORDER で指定された変数の添え字の数をn個とします

②$ORDER 関数は左から n-1 個目までの添え字が，指定された変数と同一の変数（すなわち兄弟）を調べます

③そのうちで，一番右の添え字（兄弟を区別する添え字）が，指定された変数より大きい変数（「兄」にあたる変数）がいくつかあります

④さらにそのうちで，最小の添え字をもつ変数（すぐ上の「兄」）を捜します

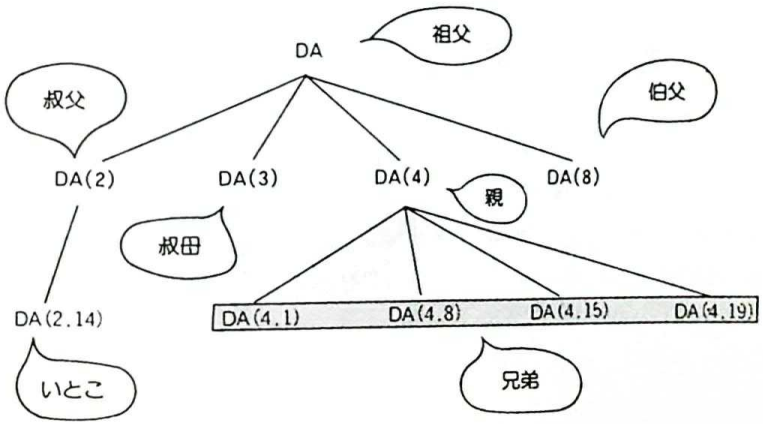
⑤捜し出した変数の一番右の添え字の値がその $ORDER の値になります

⑥もし兄弟にあたる変数がないか，あっても指定された変数より小さい添え字の変数しかない場合は，$ORDER 関数の値は空文字列になります

もうひとつ注意することがあります。$ORDER 関数の中で指定する変数はそれ自身が存在しなくても（$DATA の値が0でも）エラーにはなりません。

```
>W $O(DA(6))↵          ：DA(6)は存在しません
8
>W $O(DA(7))↵          ：これも存在しない変数です
8                        上の例と同じ値が返ることに注意
>W $O(DA(8,1))↵
6
```

fig 9-5　DA(4,の家系図



```
>W $O(DA(8,7))↵
9
>W $O(DA(8,20)↵)
                        ：エラーにはならず，空文字列が返ります
>W $O(DA(10))↵
                        ：同上
>
```

$ORDER 関数についても，いろいろな場合について何回も確かめるのが理解の早道です。

- $ORDER関数は添え字が複数あるときは，「兄弟」についてのみ検索する
- $ORDER関数は指定された変数が存在しなくてもエラーにはならない

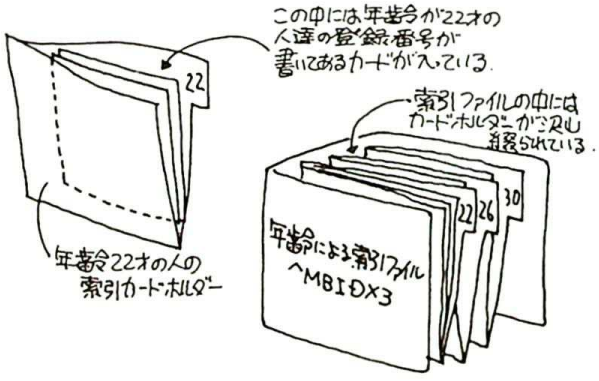
8,同一年齢の人を区別する

さて，本題にもどります。同一年齢の人を区別した索引として

```
^MBIDX3(TOSHI,NO)=""
```

という構造のグローバル変数を作ります。この構造を用いると，同一年齢の複

fig 9-6　索引カードホルダー



数の人の登録番号を管理することができます。今の例で，登録番号6と7の人が22才なので

```
^MBIDX3(22,6)
^MBIDX3(22,7)
```

という2つのグローバル変数がセットできます。登録番号10, 15, 23, 38の人もみな22才であったとすると，さらに4つのグローバル変数

```
^MBIDX3(22,10)
^MBIDX3(22,15)
^MBIDX3(22,23)
^MBIDX3(22,38)
```

がセットされるはずです。これなら同一年齢の人が何人いても問題ありません。

これは，索引カードの例で考えると，同じ索引をもつ登録番号のカードをまとめてひとつのホルダーに入れて管理することです（fig9-6)。

ところで，各変数に入っているデータは「空文字列」でこのデータ自身には意味がありません。むしろ変数の添え字に意味があります。なぜこのようなことをするのか，ピンとこないかもしれませんが，これも MUMPS プログラムのテクニックのひとつです。このまま覚えてください。

次にこれをセットするルーチンを示します。

list 9-8

```
MB054   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1985.3.22;ｷﾝﾚｲ ﾆ ﾖﾙ ｻｸｲﾝ ﾌｧｲﾙ ^MBIDX3 ﾖ ｻｸｾｲ
        ;
HAJIME  K ^MBIDX3 S NO=""
        F QQ=0:0 S NO=$O(^MBM(NO)) Q:NO="" S TOSHI=^MBM(NO,2),^MBIDX3(TOSHI,NO)
     -->=""
        Q
```

行HAJIME＋1のSET命令の^MBIDX3以下が少し変わっているだけです。

```
S ^MBIDX2(TOSHI)=NO    →    S ^MBIDX3(TOSHI,NO)=""
```

データを書き出すルーチンは少し複雑になります。ある年齢について同一年齢の人が何人登録されているかわからないので，それらを $ORDER 関数で捜さなくてはならないからです。

list 9-9

```
MB055   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1985.3.22;ﾈﾝﾚｲ ｼﾞｭﾝ ﾉ ﾃﾞｰﾀ ﾘｽﾄ
        ;
A       S TOSHI=""
        F QQ1=0:0 S TOSHI=$O(^MBIDX3(TOSHI)) Q:TOSHI="" D B
        Q                    ①               ②      ③
B       S NO=""
        F QQ2=0:0 S NO=$O(^MBIDX3(TOSHI,NO)) Q:NO="" D HYOJI
        Q                      ④               ⑤      ⑥
HYOJI   W !,NO,?5,^MBM(NO,0),?20,^MBM(NO,1),?25,^MBM(NO,2),?30,^MBM(NO,3)
        Q
```

行Aから A+2 と，行Bから B+2 が同じ形をしていることに注意してください。どちらも FOR 命令，$ORDER 関数，QUIT 命令，DO 命令が同じ形で現れています。

このルーチンの働き方は次のとおりです。

A) $ORDER 関数で最小の年齢を求めるため，年齢 TOSHI を空文字列にします

・①から③までの処理を繰り返します

① 次の年齢 TOSHI を $ORDER 関数で求めます

② もし年齢 TOSHI が空文字列ならここで繰り返しは終了します

③ 求めた年齢についてBの処理をします

B) $ORDER 関数を用いて，A から渡された年齢についての最小の登録番号を求めるため，登録番号 NO を空文字列にセットします

・④から⑥までの処理を繰り返します

④ 次の登録番号 NO を $ORDER 関数で求めます

⑤ もし登録番号 NO が空文字列ならここで繰り返しは終了

⑥ 求めた登録番号について HYOJI の処理をします

HYOJI）Bから渡された登録番号 NO の人のデータを表示します

この作業は，人間の手作業で言えば次のような処理と同じです。

① 索引カードのホルダーを年齢順にひとつ1つ取り出します。
これは $O(^MBIDX3(TOSHI)) の値を求めることに相当し，行A+1の作業にあたります。

② 取り出したホルダーには同一年齢の人のカードが何枚か入っています。こ

の中のカード一枚ごとに，そこに書いてある登録番号を調べます。

これは $O(^MBIDX3(TOSHI, NO)) の値を求めることに相当します。行B＋1の作業にあたります。

③ 求めた登録番号について，個人台帳からその登録番号の人のデータを書き写します。これは行 HYOJI にあたります。

以上の方法は，MUMPS を用いてデータの並べ替えるときの基本なので十分理解してください。

> • ^GLOBAL(データ，登録番号)=""という構造のグローバル変数を索引に用いると，同一値をもつデータが複数ある場合もそのデータ項目で並べ替えできる

9,同姓同名のデータの並べ替え

名前による並べ替えの場合でも同一年齢よりは頻度は少ないものの，同姓同名の人が登録されている可能性があります。これを処理するには年齢の場合と同様

```
^MBIDX4(NAMAE,NO)=""
```

という構造のグローバル変数を採用します。

まず，名前の索引カードを作り，同姓同名の人のカードをひとつのホルダーにまとめます。各カードには何も書き込みません（空文字列を書き込みます）。

```
K ^MBIDX4 ←―― 古いカードを捨てます
S NO=""
F QQ=0:0 S NO=$O(^MBM(NO)) Q:NO="" S NAMAE=^MBM(NO,0),^MBIDX4(NAMAE,NO)=""
```

次に索引カードのホルダーをひとつ一つ名前順に取り出します（fig 9-8）。

```
S NAMAE=""
F QQ1=0:0 S NAME=$O(^MBIDX4(NAMAE)) Q:NAMAE="" D B
```

次にその中のカードを登録番号順に1枚1枚取り出します（fig9-9）。

```
S NO=""
F QQ2=0:0 S NO=$O(^MBIDX4(NAMAE,NO)) Q:NO="" D HYOJI
```

これまでの過程をカードホルダーの例で図解すれば，fig 9-7 から fig 9-9 のとおりです。

fig 9-7

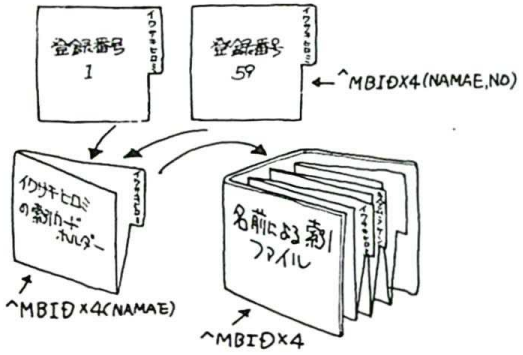


fig 9-8

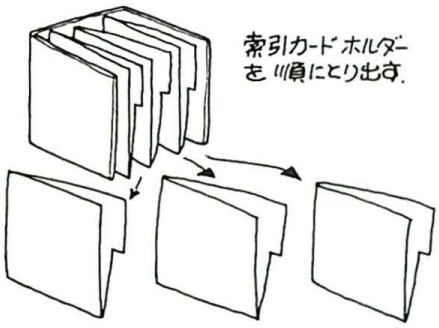
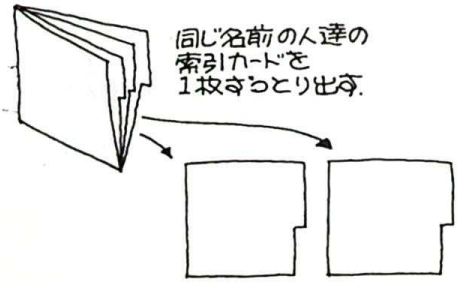


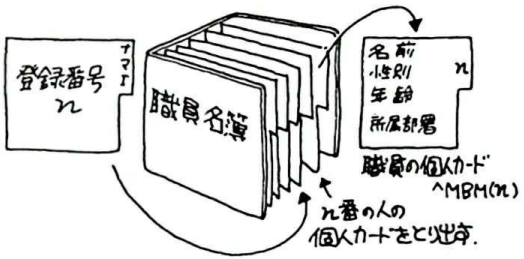
fig 9-9



得られた番号についてその人のデータを書きます（fig9-10）。

```
W !,NO,?5,^MBM(NO,0),?20,^MBM(NO,1),?25,^MBM(NO,2),?30,^MBM(NO,3)
```

fig 9-10



これらの短い手順をひとつのルーチンにまとめましょう。

list 9-10

```
MB056   ;YS,ショクイン・メイボ";1986.2.1;ナマエ ニ ヨル ナラベカエ
        ;
SET     K ^MBIDX4
        S NO=""
        F QQ=0:0 S NO=$O(^MBM(NO)) Q:NO=""  S NAMAE=^MBM(NO,0),^MBIDX4(NAMAE,NO)
    -->=""
A       S NAMAE=""
        F QQ1=0:0 S NAMAE=$O(^MBIDX4(NAMAE)) Q:NAMAE=""  D B
        Q
B       S NO=""
        F QQ2=0:0 S NO=$O(^MBIDX4(NAMAE,NO)) Q:NO=""  D HYOJI
        Q
HYŌJI   W !,NO,?5,^MBM(NO,0),?20,^MBM(NO,1),?25,^MBM(NO,2),?30,^MBM(NO,3)
        Q
```

このルーチンは，前節ではふたつに分けていたのを，ひとつにまとめたものです

各まとまりごとにその間に QUIT 命令を入れておくことを忘れないでください。ただし，行 SET からの３行と行Ａから２行は別のまとまりですが，このふたつは順番に実行されるので，その間には QUIT 命令を入れてはいけません。

Chapter 10

もっと便利にルーチン改造

データ登録，削除，問い合わせ，リストアップ，検索など通常のデータ管理に必要なルーチンは前章までで一応形ができました。この章ではデータ登録ルーチンを中心に，これらをより便利により実用的なものに改造します。

この章では新しい命令や関数は何も出てきません。今まで解説したものだけで，一般的な仕事の９割以上を処理できるからです。ルーチンの各部分はごく簡単な命令の集まりです。ただ命令などの組み合わせがやや複雑になります。

複雑な構造を理解するためには，「モジュール」という概念がとても便利です。2，でその解説をしますので，よく理解してください。

1,データ登録ルーチンの問題点

まず，データ登録ルーチンの改造から始めます。Chapter 6-10（リスト 6-16）で作成したデータの登録ルーチンです。

このルーチンには

list 10-1

```
MB026   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1985.3.14;ﾃﾞｰﾀ ﾆｭｳﾘｮｸ
        ;
KAISHI  F K=1:1:999999 D TOROKU I NO="*" Q
        Q
TOROKU  R !!,"ﾙﾝｺﾞｳ : ",NO
        I NO="*" Q
        R !,"ﾅﾏｴ ..... : ",NAMAE
        R !,"ｾｲﾍﾞﾂ .... : ",SEI
        R !,"ﾈﾝﾚｲ ..... : ",TOSHI
        R !,"ｼｮｿﾞｸ ﾌﾞｼｮ : ",BUSHO
KAKUNOU S ^MBM(NO,0)=NAMAE
        S ^MBM(NO,1)=SEI
        S ^MBM(NO,2)=TOSHI
        S ^MBM(NO,3)=BUSHO
        Q
```

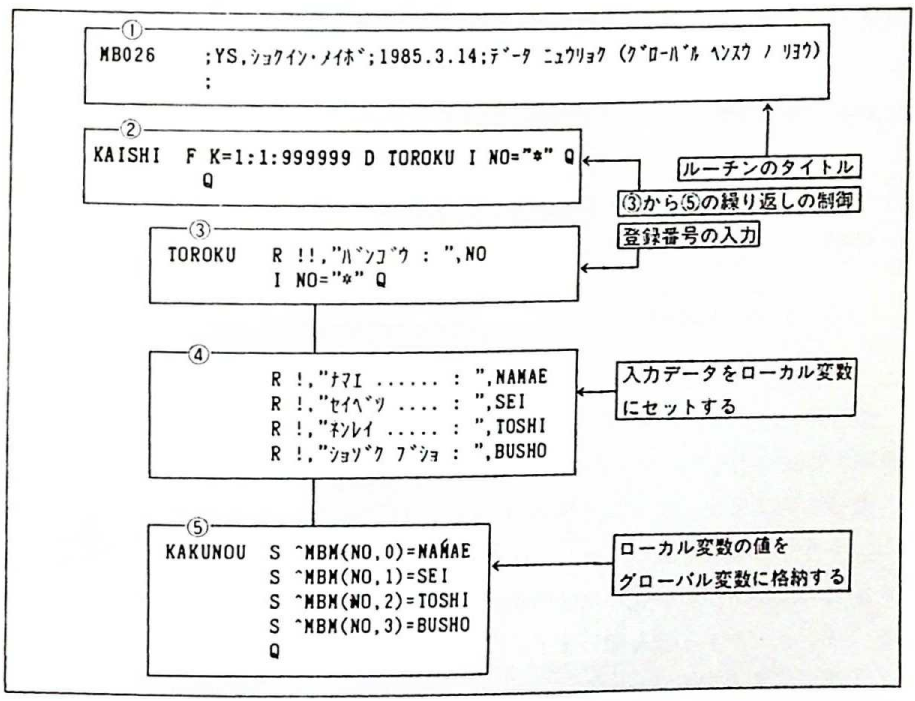
①データ入力中に，打ち間違いに気が付いても修正できない
②すでに登録してある番号で入力すると，登録してあるそのデータの上に新しいデータがかぶさり前のデータが消えてしまう

といった問題があります。これをひとつ一つ解決してみます。

2,仕事の下請け—「モジュール」

前節の登録ルーチンを改造する前に，このルーチンを4つの部分に分けて構造を考えてみましょう。

fig 10-1　ルーチンのモジュール



①のルーチンヘッダーは，「このルーチンは，何という名前で，誰が，いつ，何のために作ったか」を説明する部分です。(Chapter 3-12 参照)。

③から⑤が職員1人分のデータを登録している部分です。②はそれらの繰り返しを制御し，その働きでデータを何人分も繰り返して登録できます。

ここで，1人分のデータの登録だけに話を集中します。

1人分のデータ入力は

③データの登録番号を指定する

④データを入力し，それをローカル変数に記憶させる

⑤入力されたデータをグローバル変数に格納する

という3つの部分からできています。

ここで，構造を明確にするために③④⑤だけを取り出して，次のように変更します。

list 10-2

```
MB057   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1985.3.14;ﾃﾞｰﾀ ﾆｭｳﾘｮｸ
        ;
TOROKU  R !!,"ﾊﾞﾝｺﾞｳ : ",NO             ┐
        I NO="*" Q                       │
        D READ ──────────③'             ├─③
        D KAKUNOU ───────③"             │
        Q                                ┘
READ    R !,"ﾅﾏｴ ...... : ",NAMAE        ┐
        R !,"ｾｲﾍﾞﾂ .... : ",SEI          │
        R !,"ﾈﾝﾚｲ ..... : ",TOSHI        ├─④
        R !,"ｼｮｿﾞｸ ﾌﾞｼｮ : ",BUSHO        ┘
        Q
KAKUNOU S ^MBM(NO,0)=NAMAE               ┐
        S ^MBM(NO,1)=SEI                 │
        S ^MBM(NO,2)=TOSHI               ├─⑤
        S ^MBM(NO,3)=BUSHO               ┘
        Q
```

この例では，③の③'でデータを入力する部分④を呼びだし，③"でデータを格納する部分⑤を呼び出す形になっています。

なぜこのような一見面倒な形にするのかわからないかもしれません，しかし，もう少し解説を進めていくと徐々に理解できます。

④と⑤はひとつの独立した仕事をする単位になっています。この単位のことを「モジュール」と呼んでいます。

④はデータ入力モジュール

⑤はデータ格納モジュール

③は，具体的な作業は④⑤にまかせてしまい，ただそれらを呼び出しているだけです。他の「モジュール」を呼び出すという仕事をしているので，③のことを「他のモジュールを制御するモジュール」ということもできます。つまり

③は制御モジュール，です。

- ルーチンは一般にいくつかの仕事の単位が集まってできている
- 仕事の単位のことを「モジュール」と呼ぶ

3,入力データを確認する

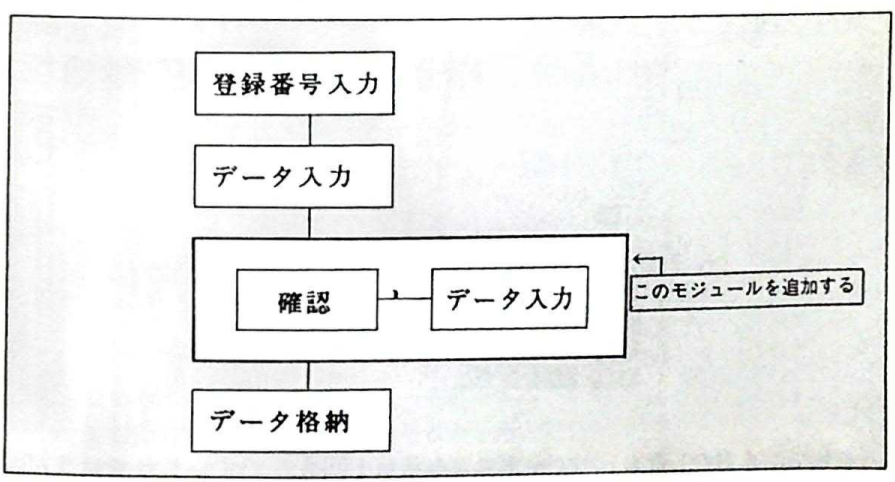
さて登録作業について考えます。

前記のルーチンでは，データを入力するとすぐにディスク上のグローバル変数に格納されてしまいます。これでは間違えて入力した場合に困ります。

入力したあと，「これでOKか？」とMUMPSがユーザーにたずねてくるようにします。そして，もしユーザーが，Y (=YES)と入力するとそのままデータを格納します。それ以外の答が入力されると，もう一度データを入力することにします。

この新しい仕事は，データ入力モジュールとデータ格納モジュールの間に入ります。

fig 10-2　モジュールの追加



リスト 10-3 では，①と②が新たに付け加わった部分です。'KAKUNIN' モジュールとそれを呼び出す DO 命令です。

'KAKUNIN' モジュールでデータ入力モジュール'READ'を呼び出してい

ることに注目してください。'READ'モジュールは，したがって制御モジュールと確認モジュールのふたつから呼び出されています。

list 10-3

```
MB058   ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1985.3.14;ﾃﾞｰﾀ ﾆｭｳﾘｮｸ
        ;
SHORI   R !!,"ﾊﾞﾝｺﾞｳ : ",NO
        I NO="*" Q
        D READ
        D KAKUNIN ←―――――― ①確認モジュールの呼び出し
        D KAKUNOU
        Q
READ    R !,"ﾅﾏｴ ..... : ",NAMAE
        R !,"ｾｲﾍﾞﾂ .... : ",SEI
        R !,"ﾈﾝﾚｲ ..... : ",TOSHI
        R !,"ｼｮｿﾞｸ ﾌﾞｼｮ : ",BUSHO
        Q
KAKUNIN R !,"* OK ﾃﾞｽｶ ? ",KOTAE
        I KOTAE="Y" Q
        D READ                    ← ②確認モジュール
        Q
KAKUNOU S ^MBM(NO,0)=NAMAE
        S ^MBM(NO,1)=SEI
        S ^MBM(NO,2)=TOSHI
        S ^MBM(NO,3)=BUSHO
        Q
```

実行します。

```
>D ^MB058↵

ﾊﾞﾝｺﾞｳ : 9↵
ﾅﾏｴ ..... : ｵｸﾀﾞ ｷｸｵ↵
ｾｲﾍﾞﾂ .... : M↵
ﾈﾝﾚｲ ..... : 27↵
ｼｮｿﾞｸ ﾌﾞｼｮ : ｼｽﾃﾑ ｶｲﾊﾂ ﾌﾞ↵
* OK ﾃﾞｽｶ ? N↵                    ：Y以外の文字を入力
ﾅﾏｴ ..... : ｵｸﾀﾞ ｷｸｵ↵
ｾｲﾍﾞﾂ .... : M↵                  ｝データの再入力
ﾈﾝﾚｲ ..... : 28↵
ｼｮｿﾞｸ ﾌﾞｼｮ : ｼｽﾃﾑ ｶｲﾊﾂ ﾌﾞ↵
>
```

これで，入力のしなおしができます。ただし1回きりです。これではまだ困るので，何回でも再入力できるようにしましょう。仕事を繰り返すときには，FOR命令を使います。

②の確認モジュールをモジュール②'で置き換えて次のようにしてください。

list 10-4

```
MB058A   ;YS,ｼｮｸｲﾝ・ﾒｲﾎﾞ;1985.3.14;ﾃﾞｰﾀ ﾆｭｳﾘｮｸ
         ;
SHORI    R !!,"ﾊﾞﾝｺﾞｳ : ",NO
         I NO="*" Q
         D READ
         D KAKUNIN
         D KAKUNOU
         Q
READ     R !,"ﾅﾏｴ ..... : ",NAMAE
         R !,"ｾｲﾍﾞﾂ .... : ",SEI
         R !,"ﾈﾝﾚｲ ..... : ",TOSHI
         R !,"ｼｮｿﾞｸ ﾌﾞｼｮ : ",BUSHO
         Q
KAKUNIN  F QQ=0:0 D KAKUNIN1 I KOTAE="Y" Q
         Q
KAKUNIN1 R !,"* OK ﾃﾞｽｶ ? ",KOTAE
         I KOTAE="Y" Q
         D READ
         Q
KAKUNOU  S ^MBM(NO,0)=NAMAE
         S ^MBM(NO,1)=SEI
         S ^MBM(NO,2)=TOSHI
         S ^MBM(NO,3)=BUSHO
         Q
```

この2行を加えます

行ラベルに注意

② 新しい確認モジュール

4,修正したい項目を指定する

まだまだ不満です。4つのデータ項目のうち，たったひとつのデータを間違えて入力しただけで，すべての項目について最初から入力しなくてはなりません。

各データ項目に番号を付けておき「OKですか？」とたずねたときに，『Y』の代わりにその番号を入力するとその項目だけ再入力できるようにしましょう。入力された番号は変数KOTAEに入っています。

変数KOTAEの内容が，1なら名前を再入力する
変数KOTAEの内容が，2なら性別を再入力する
変数KOTAEの内容が，3なら年齢を再入力する
変数KOTAEの内容が，4なら所属部署を再入力する

ようにします。このような場合分けにはIF命令を利用します。

list 10-5

```
MB059    ;YS,ｼｮｸｲﾝ・ﾒｲﾎﾞ;1985.3.14;ﾃﾞｰﾀ ﾆｭｳﾘｮｸ
         ;
SHORI    R !!,"ﾊﾞﾝｺﾞｳ : ",NO
         I NO="*" Q
         D READ
         D KAKUNIN
         D KAKUNOU
         Q
READ     R !,"1. ﾅﾏｴ ..... : ",NAMAE
         R !,"2. ｾｲﾍﾞﾂ .... : ",SEI
         R !,"3. ﾈﾝﾚｲ ..... : ",TOSHI
         R !,"4. ｼｮｿﾞｸ ﾌﾞｼｮ : ",BUSHO
         Q
KAKUNIN  F QQ=0:0 D KAKUNIN1 I KOTAE="Y" Q
         Q
KAKUNIN1 R !,"* OK ﾃﾞｽｶ ? ",KOTAE
         I KOTAE="Y" Q
         I KOTAE=1 R !,"1. ﾅﾏｴ ..... : ",NAMAE
         I KOTAE=2 R !,"2. ｾｲﾍﾞﾂ .... : ",SEI
         I KOTAE=3 R !,"3. ﾈﾝﾚｲ ..... : ",TOSHI
         I KOTAE=4 R !,"4. ｼｮｿﾞｸ ﾌﾞｼｮ : ",BUSHO
         Q
KAKUNOU  S ^MBM(NO,0)=NAMAE
         S ^MBM(NO,1)=SEI
         S ^MBM(NO,2)=TOSHI
         S ^MBM(NO,3)=BUSHO
         Q
```

場合分けです

```
>D MB059↵
ﾊﾞﾝｺﾞｳ : 10↵
1. ﾅﾏｴ ..... : ｵｵｸﾎﾞ ﾂﾈﾏｻ↵
2. ｾｲﾍﾞﾂ .... : F↵
3. ﾈﾝﾚｲ ..... : 32↵
4. ｼｮｿﾞｸ ﾌﾞｼｮ : ﾀﾞｲ1 ｹﾝｷｭｳｼｯ↵
* OK ﾃﾞｽｶ ? 2↵
2. ｾｲﾍﾞﾂ .... : M↵
* OK ﾃﾞｽｶ ? Y↵
>
```

：性別を修正したいので2と入力します
：性別だけを再入力します
：OKなのでYと入力します
：終了しました

さてこれまでのルーチンは，1人分のデータを入力するとそれで終了します。これを何回も繰り返すことができるように改造してください。

行 SHORI の前に

```
HAJIME   F QQ=0:0 D SHORI I NO="*" Q
         Q
```

という2行を挿入するだけです。

5,登録済みのデータの修正

次に，すでにディスク上に登録されているデータを修正するルーチンを考えます。これはデータ登録ルーチンを少し変更するだけで済みます。仕事としては，次のようです。

①登録番号を入力する
②その番号に，データが登録されていなければ終了
③登録されているデータをディスクから取り出す
④取り出したデータを表示
⑤修正すべき項目があれば修正する
⑥データをディスクに格納する

この仕事を何回も繰り返して行うようにしたのがリスト10-6のルーチンです。

番号をたずねたときに『＊』記号を1個入れてキーを打つと，変数NOの値は『＊』になります。すると行SHORI+1と行HAJIMEのQUIT命令が実行されて繰り返しが終了し，このルーチン全体も終わります。これもいつものパターンです。

一方，入力した番号にデータが登録されていない場合は，行SHORI+2のQUIT命令が実行され，修正作業は中断します。しかし，この場合は変数NOに何か番号が入っているので，行HAJIMEのFOR命令は終了しません。また次の番号を入力することができます。

list 10-6

```
MB060    ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1986.2.4;ﾃﾞｰﾀ ｼｭｳｾｲ
         ;
HAJIME   F QQ=0:0 D SHORI I NO="*" Q   ：このルーチン全体を繰り返します
         Q
SHORI    R !!,"ﾊﾝｺﾞｳ : ",NO            この行は繰り返しを終了するのに必要です
         I NO="*" Q
         I $D(^MBM(NO))=0 W " ｿﾉ ﾊﾝｺﾞｳ ﾊ ﾄｳﾛｸ ｻﾚﾃｲﾏｾﾝ" Q
         D TORIDASU
         D HYOJI
         D KAKUNIN
         D KAKUNOU
         Q
TORIDASU S NAMAE=^MBM(NO,0)
         S SEI=^MBM(NO,1)
         S TOSHI=^MBM(NO,2)
```

```
        S BUSHO=^MBM(NO,3)
        Q
HYOJI   W !,"1. ﾅﾏｴ ..... : ",NAMAE
        W !,"2. ｾｲﾍﾞﾂ .... : ",SEI
        W !,"3. ﾈﾝﾚｲ ..... : ",TOSHI
        W !,"4. ｼｮｿﾞｸ ﾌﾞｼｮ : ",BUSHO
        Q
KAKUNIN F QQ=0:0 D KAKUNIN1 I KOTAE="Y" Q
        Q
KAKUNIN1 R !,"* OK ﾃﾞｽｶ ? ",KOTAE
        I KOTAE="Y" Q
        I KOTAE=1 R !,"1. ﾅﾏｴ ..... : ",NAMAE
        I KOTAE=2 R !,"2. ｾｲﾍﾞﾂ .... : ",SEI
        I KOTAE=3 R !,"3. ﾈﾝﾚｲ ..... : ",TOSHI
        I KOTAE=4 R !,"4. ｼｮｿﾞｸ ﾌﾞｼｮ : ",BUSHO
        Q
KAKUNOU S ^MBM(NO,0)=NAMAE
        S ^MBM(NO,1)=SEI
        S ^MBM(NO,2)=TOSHI
        S ^MBM(NO,3)=BUSHO
        Q
```

前出のデータ登録ルーチンとこのふたつのモジュールが異なります

行HAJIMEと行KAKUNINに同じ

```
F QQ=0:0
```

というFOR命令があります。

前者はこのルーチン全体の繰り返しを管理しています。後者は，その1回の繰り返しの中で確認モジュールの繰り返しを管理しています。すなわちFOR命令の中にFOR命令が入っています。

このように重なったFOR命令が

```
F J=1:1:NN
```

のように，変数の値が重要な意味をもつ形になっているときには，変数の名前が同じにならないように注意しなくてはなりません。

たとえば

```
XXX  F J=1:1:5 D AAA
     Q
AAA  F J=1:1:5 W J
     Q
```

とすると，行XXXのFOR命令の繰り返しが1回目のとき，行AAAは繰り返しを5回実行します。その結果変数Jの値が5になります。すると行XXXのFOR命令は2回目の繰り返しのときJの値を見て，自分ももう5回繰り返した

と「誤解」し，それ以上繰り返さずにそこで終了してしまいます。その結果，D AAA という命令は１回しか実行されないことになります。

ところが，上記のルーチンでは，どちらのFOR命令も

```
F QQ=0:0
```

であり，変数QQの値はいつまでたっても０のままで，「誤解」しようがありません。したがって同じ変数を使用しても問題が生じないのです。

ひとつの慣例として，このルーチンのようにFOR命令が繰り返しのみを制御していて，変数の値や何回繰り返したかが重要でない場合には，いつもこの 'F QQ=0:0' という形を使用することにします。逆にこの命令を見たら，これは繰り返すという意味以上に何もないと考えてかまいません。

> • 繰り返しの回数や，繰り返しを制御する変数の値に重要な意味のない場合には，F QQ=D:D という形を用いることにしておくとルーチンを読むときに分かりやすく便利である

6，データの登録と修正，両用

データ登録ルーチンと，登録されているデータの修正ルーチンは，共通要素が多いので，ひとつのルーチンにまとめてしまうことができます。業務のおよその流れは以下の通りです。

次の処理を繰り返し，登録番号に『＊』記号が入力されたら処理を終わります。

①登録番号を入力する
②『＊』記号が入力されたら終了
③その番号に，データが登録されていなければデータを入力
④データが登録されていればデータをディスクから取り出し，それを表示
⑤修正すべき項目があれば修正する
⑥データをディスクに格納する

以下にルーチンのリストを掲げます。少々長くなりますが，各モジュールは今までのルーチンのコピーになっています。

list 10-7

```
MB061    ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1986.2.4;ﾃﾞｰﾀ ﾄｳﾛｸ･ｼｭｳｾｲ
         ;
HAJIME   F QQ=0:0 D SHORI I NO="*" Q
         Q
SHORI    R !!,"ﾊﾞﾝｺﾞｳ : ",NO
         I NO="*" Q
         I $D(^MBM(NO))=0 D READ
         I $D(^MBM(NO))=10 D TORIDASU,HYOJI
         D KAKUNIN
         D KAKUNOU
         Q
READ     R !,"1. ﾅﾏｴ ..... : ",NAMAE
         R !,"2. ｾｲﾍﾞﾂ .... : ",SEI
         R !,"3. ﾈﾝﾚｲ ..... : ",TOSHI
         R !,"4. ｼｮｿﾞｸ ﾌﾞｼｮ : ",BUSHO
         Q
TORIDASU S NAMAE=^MBM(NO,0)
         S SEI=^MBM(NO,1)
         S TOSHI=^MBM(NO,2)
         S BUSHO=^MBM(NO,3)
         Q
HYOJI    W !,"1. ﾅﾏｴ ..... : ",NAMAE
         W !,"2. ｾｲﾍﾞﾂ .... : ",SEI
         W !,"3. ﾈﾝﾚｲ ..... : ",TOSHI
         W !,"4. ｼｮｿﾞｸ ﾌﾞｼｮ : ",BUSHO
         Q
KAKUNIN  F QQ=0:0 D KAKUNIN1 I KOTAE="Y" Q
         Q
KAKUNIN1 R !,"* OK ﾃﾞｽｶ ? ",KOTAE
         I KOTAE="Y" Q
         I KOTAE=1 R !,"1. ﾅﾏｴ ..... : ",NAMAE
         I KOTAE=2 R !,"2. ｾｲﾍﾞﾂ .... : ",SEI
         I KOTAE=3 R !,"3. ﾈﾝﾚｲ ..... : ",TOSHI
         I KOTAE=4 R !,"4. ｼｮｿﾞｸ ﾌﾞｼｮ : ",BUSHO
         Q
KAKUNOU  S ^MBM(NO,0)=NAMAE
         S ^MBM(NO,1)=SEI
         S ^MBM(NO,2)=TOSHI
         S ^MBM(NO,3)=BUSHO
         Q
```

新規登録のとき

登録データの修正のとき

```
>D ^MB061↵

ﾊﾞﾝｺﾞｳ : 12↵
1. ﾅﾏｴ ..... : ｺｸﾗ ﾉﾌﾞｺ↵
2. ｾｲﾍﾞﾂ .... : F↵
3. ﾈﾝﾚｲ ..... : 24↵
4. ｼｮｿﾞｸ ﾌﾞｼｮ : ﾄｳｼｮｼﾂ↵
* OK ﾃﾞｽｶ ? Y↵
```

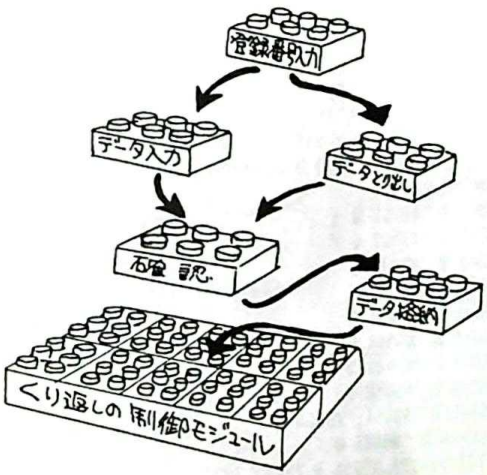
：番号12の人を新規登録します

```
ﾊﾞﾝｺﾞｳ : 10↵
1. ﾅﾏｴ ..... : ｵｵｸﾎﾞ ﾂﾈﾏｻ
2. ｾｲﾍﾞﾂ .... : M                    : 番号10の人はすでに登録してあった
3. ﾈﾝﾚｲ ..... : 32                     ので，そのデータが表示されます
4. ｼｮｿﾞｸ ﾌﾞｼｮ : ﾀﾞｲ1 ｹﾝｷｭｳｼﾂ
* OK ﾃﾞｽｶ ? 3↵
3. ﾈﾝﾚｲ ..... : 33↵
* OK ﾃﾞｽｶ ? Y↵
>
```

このように，ルーチンをモジュールに分けて考えると，モジュールを追加したり入れ換えるだけで，ごく簡単なものから，複雑なルーチンまで容易に改造することができます。これはあたかもプラスチックのブロックでおもちゃの家を作り，その窓や扉を付け替えて，別の家にしてしまうようなものです。

7,メインドライバールーチン

いままでデータ登録/修正，削除，リスト，検索などのルーチンをいくつか作成してきました。実際にこれらを利用しようとすると，ルーチン名をすぐに思い出すことはなかなか困難です（そくざにデータ削除のルーチンの名前を指摘できますか？）。



ここで，各業務のルーチンを呼び出すためのルーチンを作ることにしましょう。それを実行し，どの業務をしたいのか教えると，自動的に指定した業務を処理するルーチンを呼び出してくれるというルーチンです。

いま職員名簿のデータ処理のために，MB00 というルーチンを作ります。ユーザーはこのルーチン名だけを覚えておいて

```
>D ^MB00↵
```

とすれば，業務ごとのルーチンが自動的に呼び出されます。あとは呼び出された各ルーチンの指示に従えば職員名簿に関する作業をすべて行うことができます。

これがあれば，数多くのルーチン名を記憶しておく必要がなくなり，とても便利です。このような業務の「交通整理」をしてくれるルーチンのことを「メインドライバー」と呼んでいます。

list 10-8

```
MB00    ;YS,ｼｮｸｲﾝ･ﾒｲﾎﾞ;1986.2.18;ﾒｲﾝ･ﾄﾞﾗｲﾊﾞ
        ;
HAJIME  W !!,"*** ｼｮｸｲﾝ･ﾒｲﾎﾞ ***",!
        F QQ=0:0 D SHORI I BANGO="*" Q
        W !!,"*** ｵﾜﾘ ***",!
        Q
SHORI   W !!,"1. ﾃﾞｰﾀ ﾄｳﾛｸ ｼｭｳｾｲ"
        W !,"2. ﾃﾞｰﾀ ｻｸｼﾞｮ"
        W !,"3. ｾﾝｲﾝ ﾉ ﾃﾞｰﾀ ﾉ ﾘｽﾄ"
        W !,"4. ﾅﾏｴ ﾆ ﾖﾙ ｹﾝｻｸ"
        W !,"5. ﾀﾞﾝｾｲ ﾉﾐ ﾉ ﾃﾞｰﾀ ﾘｽﾄ"
        W !,"6. ｼﾞｮｾｲ ﾉﾐ ﾉ ﾃﾞｰﾀ ﾘｽﾄ"
        W !,"7. ｼﾃｲ ｻﾚﾀ ﾈﾝﾚｲ ﾐﾏﾝ ﾉ ﾃﾞｰﾀ ﾘｽﾄ"
        W !,"8. ﾅﾏｴ ﾆ ﾖﾙ ﾅﾗﾍﾞｶｴ"
        W !,"9. ﾈﾝﾚｲ ﾆ ﾖﾙ ﾅﾗﾍﾞｶｴ"
        W !,"10. ﾍｲｷﾝ ﾈﾝﾚｲ ﾉ ｹｲｻﾝ"
RBANGO  R !!,"* ﾄﾞﾉ ｷﾞｮｳﾑ ｦ ｼﾏｽｶ ? ",BANGO
        I BANGO="*" Q
        I BANGO=1 D ^MB061 Q
        I BANGO=2 D ^MB027 Q
        I BANGO=3 D ^MB039 Q
        I BANGO=4 D ^MB044 Q
        I BANGO=5 D ^MB041 Q
        I BANGO=6 D ^MB040 Q
        I BANGO=7 D ^MB043 Q
        I BANGO=8 D ^MB056 Q
        I BANGO=9 D ^MB054,^MB055 Q
        I BANGO=10 D ^MB046 Q
        W !,"??? ｿﾉ ﾖｳﾅ ｷﾞｮｳﾑ ﾊ ｱﾘﾏｾﾝ ???"
        Q
```

①

②

このルーチンは①で選択できる業務内容をリストし，行RBANGOでどの業務をするかを選びます。業務を選ばずに空文字列を入力するとすべての業務が終了してしまいます。②では選ばれた業務を実行するルーチンを呼び出しています。もし存在しない業務番号を入力すると，「そのような業務はありません」というエラーメッセージが表示されます。

ここで，D ^ルーチン名という命令に注意してください。たとえば行RBANGO+3は

```
I BANGO=1 D ^MB061 Q
```

となっています。これは「変数BANGOの値が1ならばルーチンMB061を実行しなさい」という意味になります。このようにルーチンの中のDO命令でも，『^』記号を付けることによってディスクに登録されているルーチンを呼び出すことができます。もともと

```
D ABC
```

はそのルーチンの中のABCという行からQUIT命令までを実行しなさいという意味でしたが

```
D ^ABC
```

はディスクに登録されている ABC というルーチンを最初から実行しなさいという意味になります。

- ディスクに登録されているルーチンを実行するには，
  D ^ルーチン名
  とする，この命令はルーチンの中でも用いることができる
- メインドライバールーチンを作っておくと便利である

8,職員名簿管理システムのまとめ

これまで職員名簿管理のルーチンをいくつも作ってきました。さらに使いやすくするには改良すべき点がありますが，何とか使えるレベルには達していると思います。これ以上の工夫は読者諸兄に任せるとして，このあたりでひとつのステップの終了とします。

以下に，ルーチンリストの例を10種掲げておきます。今までは，ルーチンの入力の便を図るため，メッセージをすべてカナにしてきましたが，ここに掲げるルーチンは参考のために漢字のメッセージを用いておきます。漢字の入力の方法については「file 3-1」を参照してください。

この職員名簿管理システムは，分かりやすくするため，データ項目を

名前　性別　年齢　所属部署

の4つに限ってきました。この項目数を増やせばもっと別の仕事に使用することができます。一般に fig 10-3 のような2次元の表で表されるデータについてはこのシステムをそのまま応用することができます。たとえば，住所録，電話番号簿，文献検索，出納簿等々，数えあげればきりがありません。いろいろと工夫してください。

fig 10-3　2次元データ表

登録番号	データ1	データ2	データ3	・・・・	データn
1				・・・・	
2				・・・・	
3				・・・・	
：	：	：	：	・・・・	：
m				・・・・	

list 10-9 ①メインドライバー

```
MB      ;YS,職員名簿管理システム;1985.3.31;ﾒｲﾝ･ﾄﾞﾗｲﾊﾞ
        ;
HAJIME  W !!,"*** 職員名簿管理システム ***",!
        F QQ=0:0 D SHORI I BANGO="" Q
        W !!,"*** 職員名簿管理システム終了 ***  ごくろうさまでした !!",!
        Q
SHORI   D HYOJI
        D RBANGO
        I BANGO="" Q
        D SHIGOTO
        Q
HYOJI   W !!,"1. データ登録・修正"
        W !,"2. データ削除"
        W !,"3. 名前による検索"
        W !,"4. 全データのリスト"
        W !,"5. 性別のデータリスト"
        W !,"6. 指定年齢未満のデータリスト"
        W !,"7. 名前による並べ替え"
        W !,"8. 年齢による並べ替え"
        W !,"9. 平均年齢の計算"
        Q
RBANGO  S CTRL=0
        F QQ=0:0 D RBANGO1 I CTRL=1 Q
        Q
RBANGO1 R !!,"* どの業務をしますか？ ",BANGO
        I BANGO="" S CTRL=1 Q
        I BANGO>0,BANGO<10 S CTRL=1 Q
        W "  ??? そのような業務番号はありません ???"
        Q
SHIGOTO I BANGO=1 D ^MBTORK Q
        I BANGO=2 D ^MBSKJO Q
        I BANGO=3 D ^MBKNSK Q
        I BANGO=4 D ^MBLALL Q
        I BANGO=5 D ^MBLSEX Q
        I BANGO=6 D ^MBLAGE Q
        I BANGO=7 D ^MBSTNAM Q
        I BANGO=8 D ^MBSTAGE Q
        I BANGO=9 D ^MBCSA Q
        Q
```

list 10-10 ②平均年齢の計算

```
MBCSA   ;YS,職員名簿管理システム;1986.2.13;平均年齢の計算
        ;
HAJIME  S GOKEI=0
        S NO=""
        F NINZU=0:1 S NO=$O(^MBM(NO)) Q:NO=""  S GOKEI=GOKEI+^MBM(NO,2)
        W !,"登録人数 = ",NINZU," 人"
        W !,"平均年齢 = ",GOKEI/NINZU," オ"
        Q
```

list 10-11 ③データ登録・修正

```
MBTORK   ;YS,職員名簿管理システム;1986.2.4;データ登録・修正
         ;
HAJIME   F QQ=0:0 D SHORI I NO="" Q
         Q
SHORI    R !!,"登録番号 : ",NO
         I NO="" Q
         I $D(^MBM(NO))=0 D READ
         I $D(^MBM(NO))=10 D TORIDASU,HYOJI
         D KAKUNIN
         D KAKUNOU
         Q
READ     R !,"1. 名前 ... ? ",NAMAE
         R !,"2. 性別 ... ? ",SEI
         R !,"3. 年齢 ... ? ",TOSHI
         R !,"4. 所属部署 ? ",BUSHO
         Q
TORIDASU S NAMAE=^MBM(NO,0)
         S SEI=^MBM(NO,1)
         S TOSHI=^MBM(NO,2)
         S BUSHO=^MBM(NO,3)
         Q
HYOJI    W !,"1. 名前 ... : ",NAMAE
         W !,"2. 性別 ... : ",SEI
         W !,"3. 年齢 ... : ",TOSHI
         W !,"4. 所属部署 : ",BUSHO
         Q
KAKUNIN  F QQ=0:0 D KAKUNIN1 I KOTAE="Y" Q
         Q
KAKUNIN1 R !,"* ＯＫですか ? ",KOTAE
         I KOTAE="Y" Q
         I KOTAE=1 R !,"1. 名前 ... ? ",NAMAE
         I KOTAE=2 R !,"2. 性別 ... ? ",SEI
         I KOTAE=3 R !,"3. 年齢 ... ? ",TOSHI
         I KOTAE=4 R !,"4. 所属部署 ? ",BUSHO
         Q
KAKUNOU  S ^MBM(NO,0)=NAMAE
         S ^MBM(NO,1)=SEI
         S ^MBM(NO,2)=TOSHI
         S ^MBM(NO,3)=BUSHO
         Q
```

list 10-12 ④データ削除

```
MBSKJO   ;YS,職員名簿管理システム;1985.3.15;データ削除
         ;
KAISHI   W !!,"*** データ削除業務 ***",!
         F QQ=0:0 D RBANGO I NO="" Q
         W !,"*** 削除  終わり ***"
         Q
RBANGO   R !,"削除する登録番号 ? ",NO
         I NO="" Q
         I $D(^MBM(NO))=0 W "  ??? その番号は登録されていません ???" Q
```

```
HYOJI   W !,"名前 ... = ",^MBM(NO,0)
        W !,"性別 ... = ",^MBM(NO,1)
        W !,"年齢 ... = ",^MBM(NO,2)
        W !,"所属部署 = ",^MBM(NO,3)
KAKUNIN R !,"本当に削除しますか ？ ",KOTAE
SAKUJO  I KOTAE="Y" K ^MBM(NO) W !,"削除しました"
        Q
```

list 10-13 ⑤名前による検索

```
MBKNSK  ;YS,職員名簿管理システム;1985.3.21;名前による検索
        ;
HAJIME  W !!,"*** 名前による検索 ***",!
        R !,"名前 ？ ",NAMAE
        W !,"番号",?6,"名前",?20,"性別",?25,"年齢",?30,"所属部署"
        S NO=""
        F QQ=0:0 S NO=$O(^MBM(NO)) Q:NO=""  I ^MBM(NO,0)[NAMAE D HYOJI
        W !,"*** 検索終わり ***"
        Q
HYOJI   W !,NO,?5,^MBM(NO,0),?20,^MBM(NO,1),?25,^MBM(NO,2),?30,^MBM(NO,3)
        Q
```

list 10-14 ⑥全データのリスト

```
MBLALL  ;YS,職員名簿管理システム;1985.3.21;全登録データのリスト
        ;
HAJIME  W !!,"*** 全登録データのリスト ***",!
        W !,"番号",?6,"名前",?20,"性別",?25,"年齢",?30,"所属部署"
        S NO=""
        F QQ=0:0 S NO=$O(^MBM(NO)) Q:NO=""  W !,NO,?5,^MBM(NO,0),?20,^MBM(NO,1),
    -->?25,^MBM(NO,2),?30,^MBM(NO,3)
        W !,"*** リスト終わり ***"
        Q
```

list 10-15 ⑦性別のデータリスト

```
MBLSEX  ;YS,職員名簿管理システム;1985.3.21;性別によるデータリスト
        ;
HAJIME  W !!,"*** 性別によるデータリスト ***",!
        R !,"M=男, F=女 ？ ",SEI
        W !,"番号",?6,"名前",?20,"性別",?25,"年齢",?30,"所属部署"
        S NO=""
        F QQ=0:0 S NO=$O(^MBM(NO)) Q:NO=""  I ^MBM(NO,1)=SEI D HYOJI
        W !,"*** リスト終わり ***"
        Q
HYOJI   W !,NO,?5,^MBM(NO,0),?20,^MBM(NO,1),?25,^MBM(NO,2),?30,^MBM(NO,3)
        Q
```

list 10-16 ⑧指定年齢未満のデータリスト

```
MBLAGE  ;YS,職員名簿管理システム;1985.3.21;指定年齢未満のデータリスト
        ;
HAJIME  W !!,"*** 指定年齢未満のデータリスト ***",!
```

```
        R !,"何才未満の人をリストしますか ? ",TOSHI
        W !,"番号",?6,"名前",?20,"性別",?25,"年齢",?30,"所属部署"
        S NO=""
        F QQ=0:0 S NO=$O(^MBM(NO)) Q:NO=""  I ^MBM(NO,2)<TOSHI D HYOJI
        W !,"*** リスト終わり ***"
        Q
HYOJI   W !,NO,?5,^MBM(NO,0),?20,^MBM(NO,1),?25,^MBM(NO,2),?30,^MBM(NO,3)
        Q
```

list 10-17 ⑨名前による並べ替え

```
MBSNAM ;YS,職員名簿管理システム;1986.2.1;名前による並べ替え
       ;
HAJIME  W !!,"*** 名前による並べ替え ***",!
        D SETIDX
        D LIST
        W !,"*** リスト終わり ***"
        Q
SETIDX  K ^MBIDX4
        S NO=""
        F QQ=0:0 S NO=$O(^MBM(NO)) Q:NO=""  S NAMAE=^MBM(NO,0),^MBIDX4(NAMAE,NO)
     -->=""
        Q
LIST    W !,"番号",?6,"名前",?19,"性別",?24,"年齢",?30,"所属部署"
A       S NAMAE=""
        F QQ1=0:0 S NAMAE=$O(^MBIDX4(NAMAE)) Q:NAMAE=""  D B
        Q
B       S NO=""
        F QQ2=0:0 S NO=$O(^MBIDX4(NAMAE,NO)) Q:NO=""  D HYOJI
        Q
HYOJI   W !,NO,?5,^MBM(NO,0),?20,^MBM(NO,1),?25,^MBM(NO,2),?30,^MBM(NO,3)
        Q
```

list 10-18 ⑩年齢による並べ替え

```
MBSTAGE ;YS,職員名簿管理システム;1986.2.1;年齢による並べ替え
        ;
HAJIME  W !!,"*** 年齢による並べ替え ***",!
        D SETIDX
        D LIST
        W !,"*** リスト終わり ***"
        Q
SETIDX  K ^MBIDX3
        S NO=""
        F QQ=0:0 S NO=$O(^MBM(NO)) Q:NO=""  S TOSHI=^MBM(NO,2),^MBIDX3(TOSHI,NO)
     -->=""
        Q
LIST    W !,"番号",?6,"名前",?19,"性別",?24,"年齢",?30,"所属部署"
A       S TOSHI=""
        F QQ1=0:0 S TOSHI=$O(^MBIDX3(TOSHI)) Q:TOSHI=""  D B
        Q
B       S NO=""
        F QQ2=0:0 S NO=$O(^MBIDX3(TOSHI,NO)) Q:NO=""  D HYOJI
        Q
HYOJI   W !,NO,?5,^MBM(NO,0),?20,^MBM(NO,1),?25,^MBM(NO,2),?30,^MBM(NO,3)
        Q
```

応用 MUMPSファイル

Chapter 10までで，基礎的なデータベース操作ができるようになっていますが，これ以降はMUMPSの応用をさらに発展させるためのヒントを集めてみました。

他のデータベースシステムプログラム言語との比較にも利用してください。

file1ではMUMPSのデータベースの構造例をあげています。自然のデータ構造をどのようにMUMPSデータベースに反映するのかを考えてください。

file2ではMUMPSの強力な文字列処理機能について紹介しています。

file3ではSP-MUMPSを例に，MUMPS以外のシステムとの連携について述べています。これらは今までのMUMPSのテキストでは解説されていません。

file4では少し便利なMUMPSのユーティリティプログラムをふたつ紹介しています。

file5は標準MUMPSの文法要素の要約です。

file6はASCII文字と8ビットのカナ文字のコード表です。

file 1 データベース

あとから様々な仕事に用いるために，大量のデータを蓄えたものを「データベース」と呼んでいます。MUMPS は，この「データベース」を作り，管理するために設計されたシステムです。したがって，様々なデータを蓄えることができます。

Chapter 10 までで解説した「職員名簿」も簡単なデータベースの 1 例です。MUMPS を用いると多種多様なデータベースを作ることができます。

ここでは，MUMPS を用いて作るデータベースの例をいくつか紹介します。

1 住所録

住所録は職員名簿と同じような構造で実現できます。

```
^JUSHO(登録番号,0)=名前（漢字）
^JUSHO(登録番号,1)=名前（かな）
^JUSHO(登録番号,2)=勤務先住所
^JUSHO(登録番号,3)=勤務先の名前
^JUSHO(登録番号,4)=勤務先の電話番号
^JUSHO(登録番号,5)=自宅住所
^JUSHO(登録番号,6)=自宅電話番号
```

データの登録／修正／検索／リストなどの作業をするルーチンは，「職員名簿」とほとんど同じものを流用できます。名前を登録するために「漢字」と「かな」のふたつの項目を設けました。これは，手紙のラベルやデータのリストには漢字の名前が必要である一方，検索や並べ替えはかなの名前で行った方が便利だからです。

各項目のデータは，アルファベット，数字，カタカナなどの 1 バイト文字[1]の場合，最大255文字までに納まらなくてはなりません。

漢字などの全角文字[2]の場合は MUMPS によって異なります。

SP-MUMPS の場合，全角文字を 2 文字分として計算します。[3]

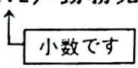
したがって，全角文字のみからなるデータでは最大127文字までで，1 バイト文字と全角文字が混在する場合は127と255の間になります。

MUMPS はデータひとつ一つについて必要最小限の記憶領域を確保します。このためデータの最大長は制限されますが，この制限内であればデータの長さを意識する必要はありません。短くても記憶領域が無駄になることはないからです。

これは MUMPS の大きな長所といえます。

ところで，もしかすると勤務先の都道府県名で検索したいとか，勤務先の部署や，内線番号を登録したいという要求があとから出てくるかもしれません。その場合には，添え字を追加します。添え字には，小数も使用できます。

```
^JUSHO(登録番号,2.1)=勤務先住所の都道府県名
^JUSHO(登録番号,3.1)=勤務先の部署名
^JUSHO(登録番号,4.1)=勤務先の内線番号
^JUSHO(登録番号,4.2)=勤務先の時間外の電話番号
```



データ登録プログラムにこれらの新しい項目を入力する部分を付け加えてください。前に入力したデータはそのままで，必要な人に関してだけ，これらの新しい項目を追加入力することができます。

MUMPS は，登録されたデータのみを記憶します。ある人については全項目のデータが登録されており，一方ある人には名前と電話番号のデータしか登録されていないかもしれません。しかし，こういう場合でもデータの長さと同様，空き領域が無駄になることを心配する必要はありません。

ただしデータを取り出すとき，つねにその添え字の変数があるかどうかを $DATA 関数で調べる必要があります。

■notes

1) 「バイト」はコンピュータで用いる情報の単位。普通，1バイトは2進数で8桁。そして，普通のアルファベット，数字，カタカナはコンピュータ内部で1文字あたり1バイトで表現されます。
2) 漢字や，漢字と同じ大きさの文字を「全角文字」と呼びます。全角文字のなかにもアルファベット，数字，カタカナなどがありますが，普通1バイト文字よりも表示したときの大きさが大きいので区別できます
3) SP-MUMPS は MS-DOS の上に乗っています。MS-DOS では全角文字はコンピュータの内部では，1文字2バイトで表現されています。このためSP-MUMPS でも全角文字は1バイト文字の2倍として計算します。

登録されていないデータを参照しようとすると「〈UNDEF〉エラー」になるので注意してください。

例：

```
I $D(^JUSHO(NO,4.2)) W !,"時間外の電話番号 = ",^JUSHO(NO,4.2)
```

2 会費納入状況の記録

テニスの同好会の会員名簿があったとします。

```
^KAIIN(会員番号,"ｶﾝｼﾞﾅﾏｴ")   =名前（漢字）
^KAIIN(会員番号,"ｶﾅﾅﾏｴ")     =名前（かな）
^KAIIN(会員番号,"ｾｲﾍﾞﾂ")     =性別
^KAIIN(会員番号,"ｾｲﾈﾝｶﾞｯﾋﾟ")=生年月日
^KAIIN(会員番号,"ｼﾞｭｳｼｮ")     =連絡先住所
^KAIIN(会員番号,"ﾃﾞﾝﾜ")       =連絡先電話番号
^KAIIN(会員番号,"ｸﾗｽ")        =テニスの腕前
```

このグローバル変数は，データ項目を区別する添え字として，文字列添え字を使用しています。文字列添え字を用いると，データとその項目番号を対応づける必要がないので管理が楽になります。

しかし反面添え字を格納する記憶領域が若干大きくなるので，大規模なデータベースには向きません。

さて，この会員名簿に会費納入状況のデータを加えることにします。

```
^KAIIN(会員番号,"ｶｲﾋ",年月)=納入年月日
```

たとえば，会員番号59番の人が，1986年1月分の会費を1985年12月23日に納入すると

```
^KAIIN(59,"ｶｲﾋ",8601)=851223
```

とセットすることができます。

会費納入データの登録プログラムの基本は次の通りです。

list 1-1

```
R !,"会員番号  ? ",NO
R !,"何年何月の会費ですか ? ",YYMM
R !,"会費を納入した年月日は ? ",HIZUKE
S ^KAIIN(NO,"ｶｲﾋ",YYMM)=HIZUKE
```

③ 文献管理

研究者にとって，文献を管理することは重要な仕事です。

コンピュータを用いた文献情報の管理検索システムがいくつも発表されていますが，よく調べてみるとあらゆる要求に答えてくれるシステムは少ないようです。常に問題になるのは，ひとつの文献について，著者が複数いたり，その文献を検索する「キーワード」が複数あることです。それも著者が10人以上いることさえあります。

1文献あたりについて，決まった量の情報を扱うようなシステムでは，これらの複数データをいくつまで登録できるようにするかが難しいところです。

著者を10人登録できるようにすると，大半の文献は1人から3人しか著者がいないので7人以上の記憶領域が無駄になります。一方5人までしか登録できないようにすると，6人以上の著者がいるときそのデータが入りません。

この無駄をなくすため，文献と著者を別のデータファイルとして管理する方法があります。しかし別管理の項目が多くなると，ファイル相互の関係を管理するのが大変になってきます。

MUMPS のグローバル変数は，添え字の数や値に何も制限がないので，このような複数データをいくつまで登録可能にするかという問題に頭を悩ませる必要はありません。

データ構造は次のようなものになるでしょう。

```
^BUNKEN(整理番号,0)=文献名
^BUNKEN(整理番号,1,1)=著者#1
^BUNKEN(整理番号,1,2)=著者#2
^BUNKEN(整理番号,1,3)=著者#3
        :                :
^BUNKEN(整理番号,1,n)=著者#n
^BUNKEN(整理番号,2)=雑誌名
^BUNKEN(整理番号,3)=巻号
^BUNKEN(整理番号,4)=ページ
^BUNKEN(整理番号,5)=出版年月日
^BUNKEN(整理番号,6)=出版地
^BUNKEN(整理番号,9,1)=キーワード#1
^BUNKEN(整理番号,9,2)=キーワード#2
```

```
^BUNKEN(整理番号,9,3)=キーワード#3
        :                    :
^BUNKEN(整理番号,9,m)=キーワード#m
```

キーワードや著者名から文献を検索する索引ファイルは次の通りです。

```
^CHOSHA(著者名,整理番号,n)=""        (空文字列)
^KEYWRD(キーワード,整理番号,m)=""    (空文字列)
```

著者名を登録するルーチンの基本は次の通りです。

list 1-2

```
CHOSHA  F K=1:1 W !,"著者 (",K,") ? " R CHOSHA(K) I CHOSHA(K)="" Q
        S N=K-1
        F K=1:1:N S ^BUNKEN(NO,1,K)=CHOSHA(K),^CHOSHA(CHOSHA(K),NO,K)=""
        Q
```

4 顧客管理

ある商店の顧客がいつどのようなものを購入したか，その客の家族はどのような構成かという情報を，データベース化しておくことができます。購入した商品のおよその耐用年数が分かれば，買い換え時期に重点的にセールスすることができます。また家族に進入学する人や成人式を迎える人がいれば，それに合わせてきめ細かいサービスをすることができるかもしれません。

MUMPS のグローバル変数の構造は，たとえば次のようなものになります。

```
^KOKYAKU(顧客番号,0)=顧客名
^KOKYAKU(顧客番号,1)=性別
^KOKYAKU(顧客番号,2)=生年月日
^KOKYAKU(顧客番号,3)=職業
^KOKYAKU(顧客番号,4)=住所
^KOKYAKU(顧客番号,5)=電話番号
^KOKYAKU(顧客番号,6,n,0)=家族#nの名前
^KOKYAKU(顧客番号,6,n,1)=家族#nの性別
^KOKYAKU(顧客番号,6,n,2)=家族#nの生年月日
^KOKYAKU(顧客番号,6,n,3)=家族#nの職業
^KOKYAKU(顧客番号,6,n,4)=家族#nの続柄
^KOKYAKU(顧客番号,7,m,0)=商品#mの名前
^KOKYAKU(顧客番号,7,m,1)=商品#mの商品分類
^KOKYAKU(顧客番号,7,m,2)=商品#mの購入年月日
```

性別が女で，年齢が20歳の家族（顧客本人も含める）がいる顧客名と電話番号のリストを表示するルーチンを掲げておきます。ただし年齢は，生年月日が6桁の整数で登録されているとすると

年齢＝（今日の年月日－生年月日）￥１００００

になります。たとえば，1954年7月28日生まれの人の，1986年2月5日現在の年齢は

(860205-540728)¥10000 --> 319477¥10000 --> 31

です。

『￥』は，割り算した結果の整数部分を取り出します（Chapter 4-7 参照）。

list 1-3

```
SAMPLE1 ;YS,SAMPLE;1986.2.5;顧客管理
        ;
HIZUKE  R !,"今日の日付 ? ",KYOU
START   S NO=""
        F QQ=0:0 S NO=$O(^KOKYAKU(NO)) Q:NO=""  D PROCA
        Q
PROCA   S SUMI=0
        D HONNIN I SUMI=1 Q
        D KAZOKU
        Q
HONNIN  S SEI=^KOKYAKU(NO,1)
        S UMARE=^KOKYAKU(NO,2)
        D CHECK
        Q
KAZOKU  S N=""
        F QQ=0:0 S N=$O(^KOKYAKU(NO,6,N)) Q:N=""  D PROCB Q:SUMI=1
        Q
PROCB   S SEI=^KOKYAKU(NO,6,N,1)
        S UMARE=^KOKYAKU(NO,6,N,2)
        D CHECK
        Q
CHECK   S TOSHI=KYOU-UMARE¥10000
        I SEI="F",TOSHI=20 D HYOJI S SUMI=1
        Q
HYOJI   W !,NO,?5,^KOKYAKU(NO,0),?25,^KOKYAKU(NO,5)
        Q
```

file 2 文字列処理

MUMPS の特徴的な機能は，ふたつあります。ひとつは，いままで話題の中心にしてきた大量のデータを管理できるということです。もうひとつが文字列処理という機能です。MUMPS の文字列処理機能は非常に強力で，数多いコンピュータ言語の中でも他に類を見ません。

MUMPS の持つ文字列処理機能のすべてを説明するのは本書の範囲を越えてしまいますが，そのさわりを解説しておきます。

1 文字列処理をする関数と演算子

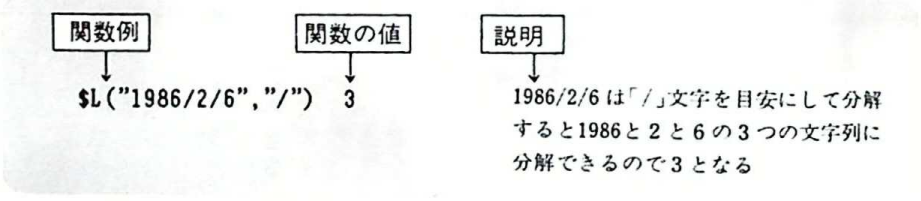
文字列処理というのは，文字のデータを連結したり，分解したり，性質を調べたりすることです。MUMPS には関数や演算子[4]としてこれらの機能が用意されています。

① $LENGTH 関数

$LENGTH 関数は，文字の数を数えます。

関数例	関数の値	説明
$L("ABC")	3	ABCは3文字
$L("A")	1	Aは1文字
$L("")	0	空文字列は0.文字

また，ある文字列を区切りにしてデータをいくつに分解できるかという数も教えてくれます。



② $EXTRACT 関数

$EXTRACT 関数は文字列からその一部を取り出します。

関数例	関数の値	説明
$E("ABCDE",1)	"A"	ABCDEの1文字目
$E("ABCDE",2)	"B"	ABCDEの2文字目
$E("ABCDE",2,3)	"BC"	ABCDEの2から3文字目
$E("ABCDE",2,4)	"BCD"	ABCDEの2から4文字目

③ $PIECE 関数

$PIECE 関数は文字列を分解します。これは MUMPS の文字列処理機能の内でももっとも強力な機能のひとつです。

関数例	関数の値	説明
$P("1986/3/16","/",1)	1986	1986/3/16 を / で区切って分解するとその1番目の文字列は 1986
$P("1986-3-16","-",2)	3	1986-3-16 を - で区切って分解するとその2番目の文字は 3
$P("1986.3.16",".",3)	16	1986.3.16 を . で区切って分解するとその3番目の文字列は 16
$P("1986-3-16","-",2,3)	3-16	1986-3-16 を - で区切って分解するとその2から3番目の文字列は 3-16

$PIECE 関数は SET 命令と組み合わせて，文字列の中に文字列を挿入することもできます。

■notes

4) 「演算子」とは，ふたつのデータの間にあって計算方法を指定する記号です。たとえば

```
3+5    3-5    X*7    GOUKEI/KAZU
```

といった式の『+，−，*，/』は，それぞれ四則演算を指定しています。これらはみな演算子です。また，データを比較する『<，>，=，[』なども演算子です。

使用例	説明
S $P(X,"/",2)=12	変数Xの値が 1986/2/8 のとき，この命令を実行すると，Xの値は 1986/12/8 になる 変数Xが定義されていないとき，この命令を実行すると，Xの値は /12 になる

④ 連結演算子

『_』記号は，連結演算子として，文字列と文字列を連結するのに用いられます。

式の例	式の値	説明
"BLACK"_"&"_"WHITE"	BLACK&WHITE	文字列の連結
100_UNIT	UNIT="m" なら 100m UNIT="光年" なら 100光年 UNIT="匹" なら 100匹	

ふたつめの例では，数値に文字列を連結していることに注意してください。

MUMPS には文字列データと数値データという区別がありません。文脈でMUMPS が適当に判断して，文字列として扱ったり，数値として計算してくれます。

たとえば

```
S X="5CATS"+"7DOGS"_"PIGS"
```

とすると，変数Xには '12PIGS' というデータがセットされます。まず加算するために '5CATS' と '7DOGS' を数値として評価し，これを加算して12とします。次にこれを文字列として評価し，'PIGS' を連結します。

⑤ パターン照合演算子

『?』記号は，文字列データのパターンを調べるのに用いることができます。この機能は MUMPS の文字列機能の中でも $PIECE 関数と並んで，もっとも強力な機能です。後述する「入力データの検査」（file 2-3 参照）などに威力を発揮します。

命令の例	説明
I X?1A W "ｱﾙﾌｧﾍﾞｯﾄ"	もしXの値がアルファベット1文字なら アルファベット と表示します
I DATA?1.5ｶ D KANADATA	もしDATAの値が1文字以上5文字以下のカナ文字ならモジュールKANADATAを実行します
I NAME?1.ｶ1" "1.ｶ W "OK"	もしNAMEの値，1文字以上のカナ，1文字の空白，1文字以上のカナならOKと表示します

『?』の右辺のAはアルファベット，カはカタカナを表します。またAの左の1や，カの左の1.5や1．は文字の数を表します。

1A　は1文字のアルファベット
1.5Aは1文字から5文字までのアルファベット
1.A　は1文字以上のアルファベット

という意味です。文字の種類を表すコードとして，他にP（記号），N（数字），U（大文字アルファベット），L（小文字アルファベット）などがあります。

2 日付の変換

文字列処理の例として日付の形式変換を考えます。

日付には，さまざまな表し方があります。たとえば次の日付はみな同じです。

```
1986.2.4
860204
4 FEB 1986
1986年2月4日
S61.2.4
```

これらの間で形式を変換するルーチン例を以下に示します。

① 860204---→1986.2.4

はじめは，860204 のふたけた数字のラレツ表示から西暦表示への変換です。以下②，③，④もそれぞれ左から右への変換方法を示します。

list 2-1

```
DTCNV1  ;YS,日付変換;1986.2.6;YYMMDD --> YYYY.MM.DD
        ;
READ    R !,"日付 (YYMMDD) ? ",A
SET     S YY=$E(A,1,2)
        S MM=$E(A,3,4)
        S DD=$E(A,5,6)
        S B=1900+YY_"."_+MM_"."_+DD
WRITE   W " ==> ",B
        Q
```

```
>D ^DTCNV1↵

日付 (YYMMDD) ? 860131↵ ==> 1986.1.31
>
```

A＝"860131" のとき MM（＝$E(A, 3, 4)）は "01" ですが，＋MM は "1" になります。

```
>W MM
01
>W +MM
1
>
```

このように，『＋』記号は余分な 0 を取り除く機能があります。

② 1986.2.4---→860204

list 2-2

```
DTCNV2  ;YS,日付変換;1986.2.6;YYYY.MM.DD --> YYMMDD
        ;
READ    R !,"日付 (YYYY.MM.DD) ? ",A
SET     S YY=$P(A,".",1)
        S MM=$P(A,".",2)
        S DD=$P(A,".",3)
        S B=$E(YY,3,4)_$E(MM+100,2,3)_$E(DD+100,2,3)
WRITE   W " ==> ",B
        Q
```

```
>D ^DTCNV2↵

日付 (YYYY.MM.DD) ? 1956.3.18↵ ==> 560318
>
```

この例では，A＝"1956. 3. 18" なので MM（$P(A, ". ", 2)）は 3 です。

```
$E(MM+100,2,3)
```

は，まず MM＋100 を計算するので

```
$E(103,2,3)
```

となります。103の２文字目から３文字目までを取り出すと03になります。

```
>W MM↵
3
>W $E(MM+100,2,3)↵
03
>
```

③ 1986.2.4---→4 FEB 1986

list 2-3

```
DTCNV3  ;YS,日付変換;1986.2.6;YYYY.MM.DD --> DD MMM YYYY
        ;
READ    R !,"日付 (YYYY.MM.DD) ? ",A
SET     S YY=$P(A,".",1)
        S MM=$P(A,".",2)
        S DD=$P(A,".",3)
        S B=DD_" "_$P("JAN/FEB/MAR/APR/MAY/JUN/JUL/AUG/SEP/OCT/NOV/DEC","/",M)_"
      --> "_YY
WRITE   W " ==> ",B
        Q
```

```
>D ^DTCNV3↵

日付 (YYYY.MM.DD) ? 1982.10.20↵ ==> 20 OCT 1982
>
```

$PC"(JAN/以下の $PIECE 関数の働きが分かれば難しくありません。

```
JAN/FEB/MAR/APR/MAY/JUN/JUL/AUG/SEP/OCT/NOV/DEC
```

という文字列は「/」記号を区切りにして分解すると

JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC
①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩	⑪	⑫

となります。この MM 番目の文字列を取り出すのですから，MM＝10 のときは，$PIECE の値は OCT になります。

$PICECE 関数を

```
$P("JANUARY/FEBRUARY/MARCH/APRIL/MAY/JUNE/JULY/AUGUST/SEPTEMBER/OCTOBER/N
OVEMBER/DECEMBER","/",MM)
```

とすれば，上記の例では，次のようになります。

```
>D ^DTCNV3↵

日付 (YYYY.MM.DD) ? 1982.10.20↵ ==> 20 OCTOBER 1982
>
```

④ S29.7.28---→1954年7月28日

list 2-4

```
DTCNV4  ;YS,日付変換;1986.2.6;ZYY.MM.DD --> YYYY年MM月DD日
        ;
READ    R !,"日付 (ZYY.MM.DD) ? ",A
SET     S Z=$E(A,1)
        S YY=$E($P(A,".",1),2,999)
        S MM=$P(A,".",2)
        S DD=$P(A,".",3)
        S YY=YY+$S(Z="M":1867,Z="T":1911,Z="S":1925)
        S B=YY_"年"_MM_"月"_DD_"日"
WRITE   W " ==> ",B
        Q
```

$SELECT関数

```
>D ^DTCNV4↵

日付 (ZYY.MM.DD) ? S20.8.15↵ ==> 1945年8月15日
>
```

この例では，$P(A,".",1) の値は S20 です。したがって $E($P(A,".",1),2,3) は，$E("S20",2,999) ということです。これは，S20 の2文字目から999文字目までを取り出すということですが，999文字まではないので，2文字目から「最後まで」を取り出すという意味になります。

$S 以下の関数は，$SELECT 関数といい，値の場合分けに用いることができます。

```
S YY=YY+$S(Z="M":1867,Z="T":1911,Z="S":1925)
```

は，Z="M" なら1867を，Z="T" なら1911を，Z="S" なら1925を YY に加えなさいという命令です。この1行は

```
I Z="M" S YY=YY+1867
I Z="T" S YY=YY+1911
I Z="S" S YY=YY+1925
```

と表現しても同じことです。

③入力データの検査

MUMPS では，データに文字列，数値等の区別はありません。

```
R !,"名前 ? ",NAMAE
```

という命令で，ユーザーがどのようなデータを入れても，MUMPS は素直にそのデータを変数 NAMAE にセットします。ユーザーは誤って性別のMを入れるかもしれないし，年齢を入れてしまうかもしれません。

個人の研究用に利用しているシステムでは間違ったデータを入力してもその人の責任なので問題になりません。しかし，多数の人が共有しているシステムや，データ入力を他の人に頼むときには誤入力は大きな問題です。

そこで，真に実用的なシステムでは，ユーザーがあり得ないデータを入力するとそれを検出し，ユーザーにそのむね教えるようにします。これによってデータの信頼度が増し，大規模なデータベースの価値も高まります。

このようなチェックには，MUMPS の文字列処理機能が役に立ちます。

以下に入力チェックの例をあげておきます。

①名前のチェック

list 2-5

```
RNAME   F CTRL=0:0 D RNAME1 Q:CTRL>0
        Q
RNAME1  R !,"名前 ? ",NAME
        I NAME?1.ｶ1" "1.ｶ S CTRL=1 Q
        W " ??? 名前の書式に合いません ???"
        Q
```

パターン照合演算子『？』を用いてデータの書式を検査します。この演算子は入力データのチェックに威力を発揮します。『？』の右辺の1.ｶ1""1.カの

1.カ は1文字以上のカナ文字

1" " は1つの空白文字

を表します。

```
I NAME?1.ｶ1" "1.ｶ S CTRL=1 Q
```

は，「変数 NAME の値が，1文字以上のカナ文字に1つの空白文字，さらに1文字以上のカナ文字が連結した形のデータなら，S CTRL=1 Q を実行し

なさい」という意味になります。

```
>D RNAME↵
名前 ? 1986.2.7↵  ??? 名前の書式に合いません ???
名前 ? JOHN LENON↵   ??? 名前の書式に合いません ???
名前 ? ﾀﾆﾄｵﾙ↵
>
```

②性別のチェック

list 2-6

```
RSEX    F CTRL=0:0 D RSEX1 Q:CTRL>0
        Q
RSEX1   R !,"性別 ? ",SEX
        I (SEX="M")!(SEX="F") S CTRL=1 Q
        W " ??? 男なら 'M', 女なら 'F' を入力してください ???"
        Q
```

```
I (SEX="M")!(SEX="F") S CTRL=1 Q
```

は,「もし SEX＝"M" または SEX＝"F" が正しければ, S CTRL＝1 Q を実行しなさい」という意味です。

『！』記号は, ここではふたつの条件のいずれかが正しいという意味になります。

「または」とか「論理和」と呼ばれます。WRITE 命令や READ 命令で改行を指定する『！』とは意味が異なります。

```
>D RSEX↵

性別 ? ｵﾄｺ↵  ??? 男なら 'M', 女なら 'F' を入力してください ???
性別 ? M↵
>
```

③コードのチェック

コンピュータにデータを入力するとき, 入力を簡略化するために数字や記号のコードを用いることがあります。入力したコードが正しいかどうかは $DATA 関数を利用すると簡単にチェックできます。

今, 商品の名前, 規格などが商品コードの下にグローバル変数 ^SHOHIN として登録されているとします。

```
^SHOHIN(CODE,0)=商品名
^SHOHIN(CODE,1)=メーカー名
      :              :
```

すると，次のようなチェックができます。

list 2-7

```
RCODE   F CTRL=0:0 D RCODE1 I CTRL>0 Q
        Q
RCODE1  R !,"商品コード ? ",CODE
        I $D(^SHOHIN(CODE)) W " ==> ",^SHOHIN(CODE,0) S CTRL=1 Q
        W "   ??? そのような商品コードは登録されていません ???"
        Q
```

```
>D RCODE↵

商品コード ? 365↵   ??? そのような商品コードは登録されていません ???
商品コード ? FD54↵  ==> ２ＤＤフロッピー・ディスク
>
```

ここで，商品コードは数値である必要がまったくないことに注目してください。

file 3 システムの連携

パーソナルコンピュータには，ワープロ，グラフ作成，表計算などの優れたプログラムがいくつも開発されています。

SP-MUMPS は MS-DOS に乗っているので，同じように MS-DOS に乗っている他のシステムとデータの交換ができます。また，バージョン1.1D以降の SP-MUMPS では $ZCALL 関数という拡張機能が付きました。これは MUMPS の中から他のシステムを呼び出すものです。ここでは，こういった他のシステムとの連携について簡単に解説します。

ここで解説する内容の多くは，SP-MUMPS特有のものです。ただし MS-DOS などの汎用 OS と呼ばれるシステムの上に乗っている MUMPS には同じような機能があります。また図形処理については，図形が表示できる端末装置があれば，他の MUMPS でも同様なことができます。

1 漢字処理

最近のパーソナルコンピュータの漢字処理機能には眼をみはるものがあります。

SP-MUMPS は MS-DOS の上に乗っており，データ入力などの作業は MS-DOS の機能を利用しています。その MS-DOS の入力プログラム自体にかな（ローマ字）漢字変換システムが組み込まれてしまうので，MUMPS でも連文節変換を簡単に利用することができます。SP-MUMPS の場合

① NEC の文節変換システム

②ATOK4[5]

③ VJE-α およびΣ[6]

などのかな（ローマ字）漢字変換システムとして使用できることが確認されています。

これらの漢字変換システムの MS-DOS への組み込み方や，変換の方法はそれぞれのマニュアルを参照してください。

```
>R !,"NAME ? ",X↵

NAME ? 奥田　規久雄↵        ：このようにREAD命令で漢字を入力する
>W X↵                          ことができます
奥田　規久雄
>
```

SP-MUMPS の場合，データや注釈文の他，ルーチン名，行ラベル名，グローバル変数名，ローカル変数名などにも漢字（を含む全角文字）を使用することができます。これらは標準 MUMPS にはない仕様ですが，うまく利用すると，日本人にとって非常に読みやすいルーチンを書くことができます。ただし，他の MUMPS にプログラムを移すことができなくなるので注意が必要です。

```
>S 年齢=25↵              ：'年齢' という変数です

>W↵

X="奥田　規久雄"
年齢="25"
>
```

次に掲げるのは，変数や行ラベルに漢字を使ったルーチンです。「職員名簿管理プログラム」のデータ登録・修正ルーチンを書き直してみました。

list 3-1

```
MB登録  ;YS,職員名簿;1986.2.4;データの登録修正
        ;
開始    F QQ=0:0 D 処理 I 番号="" Q
        Q
処理    R !!,"登録番号 : ",番号
        I 番号="" Q
        I $D(^職員名簿(番号))=0 D 入力
        I $D(^職員名簿(番号))=10 D 取出,表示
        D 確認
        D 格納
        Q
入力    R !,"1. 名前 ... : ",名前
        R !,"2. 性別 ... : ",性別
```

■notes

5) ATOK4 はジャストシステムの登録商標です。

6) VJE-αはバックス，VJE-ΣはASCIIの登録商標です。

```
        R !,"3. 年齢 ... : ",年齢
        R !,"4. 所属部署 : ",部署
        Q
取出    S 名前=^職員名簿(番号,0)
        S 性別=^職員名簿(番号,1)
        S 年齢=^職員名簿(番号,2)
        S 部署=^職員名簿(番号,3)
        Q
表示    W !,"1. 名前 ... : ",名前
        W !,"2. 性別 ... : ",性別
        W !,"3. 年齢 ... : ",年齢
        W !,"4. 所属部署 : ",部署
        Q
確認    F QQ=0:0 D 確認1 I 答="Y" Q
        Q
確認1   R !,"* OK ですか ? ",答
        I 答="Y" Q
        I 答=1 R !,"1. 名前 ... : ",名前
        I 答=2 R !,"2. 性別 ... : ",性別
        I 答=3 R !,"3. 年齢 ... : ",年齢
        I 答=4 R !,"4. 所属部署 : ",部署
        Q
格納    S ^職員名簿(番号,0)=名前
        S ^職員名簿(番号,1)=性別
        S ^職員名簿(番号,2)=年齢
        S ^職員名簿(番号,3)=部署
        Q
```

2 画面制御

MUMPS 自身には画面の全部（または一部）を消したり，文字の色を変えたり，カーソルを移動するといった画面制御の命令はありません。しかし，コンピュータのテレビ端末の多くはそのような機能を持っています。

PC-9801 のモニター画面にも画面制御機能があり，これを利用すれば，MUMPS でも画面制御が簡単にできます。

PC-9801 の MS-DOS の場合を例に掲げます。これはほぼ ANSI 規格という規格にあてはまるので，広く応用できます。

```
W *26                    ：画面を消す
W *27,"[",Y,";",X,"H"    ：カーソルを（X,Y）座標に移動する
W *27,"[0K"              ：画面のカーソルのある行の，カーソルから右端までを消す
W *27,"[0J"              ：画面のカーソルのある行から下を消す
W *27,"[33m"             ：このWRITE命令以降に表示した文字は黄色になる
W *7                     ：ピッと音を鳴らす
```

W *Xという命令は，コンピュータの内部コードXにあたる文字を出力するというものです。ただしXは10進数です。

```
>W *65↵
A                    ：アルファベットAの内部コードが 65 なのでAと表示
>                      されました
```

上記以外の制御については，MS-DOS の説明書に解説されているので，それを参照してください。

ところで，MUMPS には XECUTE 命令というデータを実行する命令があるので，次のようなことができます。

list 3-2

```
USCC    ;YS,UTILITY;1986.2.8;画面制御手順のセット
        ;
SET     S CLS="W *26"
        S SETCSR="W *27,""[",Y,";",X,""H"""     ：引用「"」の数に注意
        S EOL="W *27,""[0K"""
        S EOS="W *27,""[0J"""
        S CYELLOW="W *27,""[33m"""              ＊とYの順序に注意
        S CDFLT="W *27,""[m"""
        S BELL="W *7"
```

引用符『"』が多く感じますが，引用符の中では，引用符ふたつ『""』でひとつの引用符を表すのでこれで良いのです。

```
>S Q="""""↵          ：2つの引用符が引用符にはさまれているので，
>W Q↵                  合計4つの引用符が並んでいます
"                    ：変数Qには引用符1個がセットされています
>
```

ではこのルーチンを実行しましょう。

```
>D ^USCC↵

>W↵

BELL="W *7"
CDFLT="W *27,"[m""
CLS="W *26"
CYELLOW="W *27,"[33m""                  ：セットされている変数です
EOL="W *27,"[0K""
EOS="W *27,"[0J""
SETCSR="W *27,"[",X,";",Y,"H""
>
```

さて次の命令を確かめてください。

```
>X CLS↵
```

どうなりましたか？　画面上の文字がすべて消えて，カーソルが左上の隅に移動します。

```
>X CYELLOW↵
```

この命令を実行すると，それ以降に表示される文字が黄色くなります。

```
>X CDFLT↵
```

これで文字の色はもとに戻ります。

```
>X CLS F J=1:1:500 S X=$R(80)+1,Y=$R(24)+1 X SETCSR W J↵
```

画面が消えて，カーソルが画面のあちこちに飛び，数字がバラバラの位置に表示されるはずです。

$R は，$RANDOM 関数といい，$R(N) とすると，0 から N-1 までの乱数を発生します。X SETCSR で画面上の座標 (X,Y) にカーソルが移動し，W J でそこに数値を表示するのです。

このように，変数に制御手順をセットしておいて，XECUTE 命令でそれを実行することができます。変数に，画面制御のシンボルになるような名前を付けておくと，具体的な画面制御の手順を覚えていなくても，その名前で手順を呼び出せるので便利です。

XECUTE 命令や $RANDOM 関数について詳しくは MUMPS の他の教科書を参照してください。

つぎのリストは，常に決まった位置 (5, 10) で名前の入力を行うものです。入力データに不都合があると，ベルが鳴りエラーメッセージが画面の最下行に表示されます。

list 3-3

```
RNAME   F CTRL=0:0 D RNAME1 Q:CTRL>0
        Q
RNAME1  S X=5,Y=10 X SETCSR,EOL
        R "名前  ? ",NAME
        I NAME?1.ｶ1" "1.ｶ S CTRL=1 Q
        S X=0,Y=24 X SETCSR,BELL
        W "??? 名前の書式に合いません ???"
        Q
```

X SETCSR でカーソルを画面上の座標 (X,Y) に移動します。それ以降に

出力される文字はその位置に表示されます。

X EOL は，そのときカーソルのあった行のカーソルから右側を消します。

X BELL は，ピッと音を鳴らします。

3 図形をかく

標準 MUMPS の仕様には，図形を処理する機能は規定されていません。なかには UCD-microMUMPS[7]のように図形を描いたり，それをファイルとしてディスクに蓄える機能を持つものもありますが，SP-MUMPS には図形処理機能がありません。

しかし PC-9800 の MS-DOS 用に，EGR98[8]というプログラムが市販されているので，これを利用すると SP-MUMPS でもカラーの図を描くことができるようになります。

EGR98 を利用するには，MS-DOS を起動した状態で，MUMPS を起動する前に，EGR98 を 1 回実行します。

```
A>EGR98↵
```

すると画面に EGR98 という大きなタイトルが表示されます。そのあとで，MUMPS を起動してください。さらに

```
>W *27,*7,"INIT",!↵

>W *27,*7,"SCREEN 3,0,0,1",!↵
```

としてください。

これは図形処理の初期化の作業です。これで MUMPS で図形を描くことができるようになりました。たとえば

```
>W *27,*7,"LINE 100,100,200,200,1,2",!↵
```

とすると画面やや左上に青い正方形が描かれます。

ところで

■notes

7) EGR98 はカノープス電子の登録商標です

8) UCD-micro MUMPS は米国カリフォルニア大学デービス校で作られたソフトウェアです

```
W *27,*7
```

は EGR98 を呼び出す手順です。そしてそれに続く文字列が EGR98 に対する命令になります。最後に改行の『！』記号を忘れずに付けてください。またEGR98の機能について詳しくはEGR98のマニュアルを参照してください。

次に例として，円グラフを描くルーチンを掲げます。

このルーチンも，行ラベル，変数に全角文字を使用してみました。

標準 MUMPS の言語仕様の範囲内に納めるためには，すべてローマ字にしてください。なお「π」も全角文字の変数名です。

リスト 3-4 のルーチン中の，行「初期化」に

```
$C(27,7)
```

という関数があります。これは $CHAR 関数といって，内部コードを指定するとそれに対応する文字を発生します。たとえば

```
>S X=$C(65)↵

>W X↵
A
>
```

内部コードＸとＹの２文字を出力するには，次のような方法があります。

```
① W *X,*Y    ② W $C(X,Y)    ③ S A=$C(X,Y) W A
```

いずれも結果として同じことになります。

さらに行「座標１」に，$ZF という関数が使われています。これは数値演算の関数です。

```
$ZF("COS",X)  $ZF("SIN",X)  $ZF("TAN",X)  $ZF("LOG",X)
```

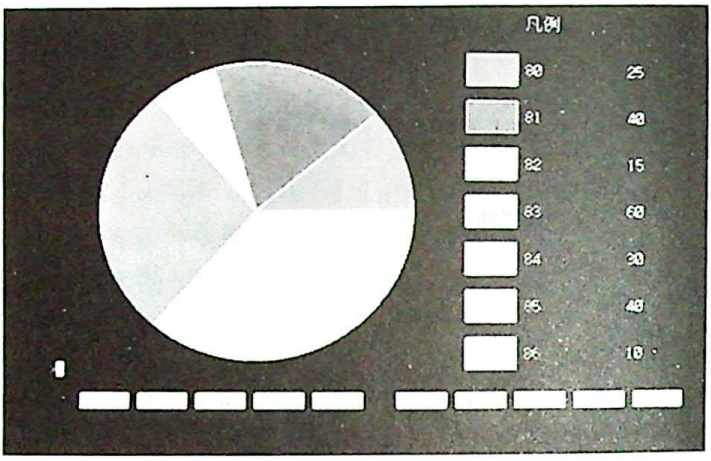
という使い方ができます。これらはそれぞれ，sin X, cos X, tan X, log X を計算してくれます。

list 3-4

```
EGR02   ;YS,SAMPLE/EGR98;1986.2.9;円グラフの例
        ;
開始    D 初期化
        D 入力
        D 計算
        D 描画
終了    Q
初期化  S G=$C(27,7)
        S π=3.141593
        W G,"INIT",!
```

```
        W G,"SCREEN 3,0,0,1",!
        W G,"CLS",!
        W *26
        Q
入力    F J=1:1 D R項目名 Q:X=""  D Rﾃﾞｰﾀ
        S 項目数=J-1
        Q
R項目名 W !,J,":",?5,"項目名 ? " R X
        I X="" Q
        S 項目名(J)=X
        Q
Rﾃﾞｰﾀ   R !?5,"ﾃﾞｰﾀ ? ",ﾃﾞｰﾀ(J)
        Q
計算    D 合計
        D 比率
        D 座標
        Q
合計    S 累積(0)=0
        F J=1:1:項目数 S 累積(J)=累積(J-1)+ﾃﾞｰﾀ(J)
        S 合計=累積(項目数)
        Q
比率    S 比率(0)=0,比率(項目数)=1
        F J=1:1:項目数-1 S 比率(J)=累積(J)/合計
        Q
座標    F J=0:1:項目数 D 座標1
        Q
座標1   S ZX(J)=$ZF("COS",2*π*比率(J))*150¥1+200
        S ZY(J)=-$ZF("SIN",2*π*比率(J))*150¥1+200
        Q
描画    W *26
        W G,"CLS",!
        D 円ｸﾞﾗﾌ
        D 凡例
        Q
円ｸﾞﾗﾌ  F J=1:1:項目数 D 円ｸﾞﾗﾌ1
        Q
円ｸﾞﾗﾌ1 W G,"CIRCLE 200,200,150,150,7,",ZX(J-1),",",ZY(J-1),",",ZX(J),",",ZY(J),
    -->",",1,!
        W G,"PAINT",ZX(J-1)+ZX(J)¥2,",",ZY(J-1)+ZY(J)¥2,",",J,",7",!
        Q
凡例    S X=58,Y=1 D SETCSR W "凡例"
        S X1=400,X2=448
        F J=1:1:項目数 S Y1=48*J-8,Y2=Y1+32 D 四角
        F J=1:1:項目数 S X=58,Y=3*J+1 D SETCSR W 項目名(J) S X=70,Y=3*J+1 D SETC
    -->SR W ﾃﾞｰﾀ(J)
        Q
四角    W G,"LINE",X1,",",Y1,",",X2,",",Y2,",",J,",2",!
        W G,"LINE",X1,",",Y1,",",X2,",",Y2,",7,1",!
        Q
SETCSR  W *27,"[",Y,";",X,"H"
        Q
```

fig 3-1
MUMPSの作図例



4 他のシステムへデータを渡す

MUMPS で作ったデータを，同じ MS-DOS 上で動く他のシステムに渡すこともできます。

たとえば，前節では EGR98 というプログラムを利用して，MUMPS 自身でグラフなどの図形をかくことを解説しました。しかし最近はグラフを描く専用プログラムがいくつも発表されており,これらを利用するのも便利な方法です。

ここでは Simple Chart [9]というグラフ作成プログラムを使用します。

Simple Chart にデータを渡すには，ASCII ファイルと呼ばれるデータファイルをディスク上に作ります。キーボードから入力されたデータを SMPLCHRT. ASC という名前の ASCII ファイルとしてディスクに書き込むルーチンをリスト 3-5 に掲げます。

O 51：…U51 W *26/C51 の命令に注目してください。

```
O 51:("A:SMPLCHRT.ASC":"W")
```

は，OPEN 命令で，ディスク装置A：に，SMPLCHRT.ASC という名前の ASCII ファイルを作ることを宣言しています。

51はホストファイルサーバー（HFS）という論理装置の番号です。この H

■notes

9） Simple Chartはステラシステムの登録商標です

HFS を用いると，あたかもモニターテレビ画面やプリンタにデータを出力するのと同じようなやり方でディスクにデータを書き込み，ASCII ファイルを作ることができます。

```
U 51
```

は，USE 命令で，これ以降は装置51番（すなわち HFS）にデータを出力することを宣言しています。

O 51だけでは，51番ドアを開いた（OPEN した）だけなので，このドアを用いて（USE して）データが出入りすることを宣言しなくてはならないのです。

USE 命令が実行されると，WRITE 命令で出力するデータはみな HFS に書き込まれます。そして，CLOSE 命令

```
C 51
```

で51番のドアを閉じて終わります。CLOSE する前に，次のようにしています。

```
W *26
```

これは，Simple Chart がこのデータを用いるときに，データはここで終わりということを認識するのに必要な文字です。

MUMPS を終了して，MS-DOS に戻り，そこから Simple Chart を呼び出します。(file 3-6 で説明する ZCALL 関数を用いると，MUMPS を終了せずに Simple Chart を呼びだすことができます)

```
A>S↵
```

Simple Chart のメニューから，まず① データ入力を選び，さらに，② ASCII ファイルを選択してください。そしてリスト 3-5 のルーチンで作ったSMPLCHRT.ASC というファイル名を指定すると，Simple Chart がそのデータを取り込んでくれます。

list 3-5

```
LSC01   ;YS.Link-Demo;1986.2.10;Simple Chart 用の ASCII ファイルを作る
        ;
START   D RDATA
        D ASCFILE
        Q
RDATA   R !,"表題 ? ",HEADER
        R !,"脚注 ? ",FTNOTE
        F J=1:1 D RCLMNAM I X="" Q
```

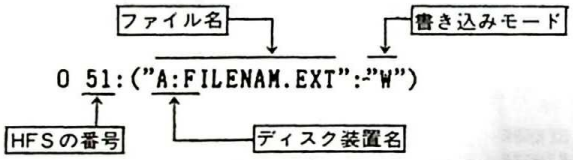
```
        S NOC=J-1
        F K=1:1 D RROW I X="" Q
        S NOR=K-1
        Q
RCLMNAM W !,"欄(",J,")の名前 ? " R X
        I X="" Q
        S CLMNAM(J)=X
        Q
RROW    W !,"列(",K,")の名前 ? " R X
        I X="" Q
        S ROWNAM(K)=X
        F J=1:1:NOC W !?5,CLMNAM(J)," のデータ ? " R DATA(K,J)
        Q
ASCFILE O 51:("A:SMPLCHRT.ASC":"W")                    :OPEN命令
        U 51                                           :USE命令
        W HEADER,",",FTNOTE,!
        W CLMNAM(1)
        F J=2:1:NOC W ",",CLMNAM(J)
        W !
        F K=1:1:NOR D WRTROW
        W *26
        C 51                                           :CLOSE命令
        Q
WRTROW  W ROWNAM(K)
        F J=1:1:NOC W ",",DATA(K,J)
        W !
        Q
```

5 ワープロと連携する

MUMPS で作成したデータを MS-DOS 上で作動するワープロに渡すこともできます。すると MUMPS で収集したデータをワープロの文章の中に織り込んだり，ワープロで表を作成して，それをデータとして MUMPS のデータベースに取り込んだりすることができます。

その方法は前節とほとんど同じです。

① HFS を OPEN する

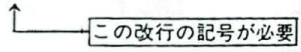


② HFS に出力することを宣言する

```
U 51
```

③データを出力する

```
W ﾃﾞｰﾀ,!
```



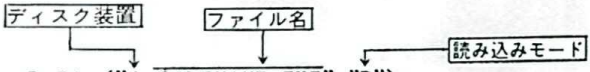
④ファイルの終了文字を書き込む

```
W *26
```

⑤ HFS を CLOSE する

```
C 51
```

こうして作成した ASCII ファイルをワープロで読み込めばそれを編集することができます。逆にワープロで作成した文章を MUMPS のデータとして取り込むこともできます。




```
O 51:("A:FILENAME.EXT":"R")
U 51
F J=1:1 R X Q:X="" S ^DATA(J)=X
C 51
```

でデータがグローバル変数 ^DATA にセットされます。ただし，このときひとつ一つのデータが 1 バイト文字（全角文字 1 文字は 2 文字分）で255文字以内になるように注意してください。さもないと，次のようなエラーになります。

```
<MXSTR> 文字列が最大長を越えました
```

ルーチンをワープロとやり取りすることもできます。ルーチンをワープロを利用して制作し，それを MUMPS が読み取って実行するとか，MUMPS で作ったルーチンをワープロに渡して修正するといったことができます。

①ルーチンを ASCII ファイルにする

```
>ZL ABC

>O 51:("A:XFER.RTN":"W")

>U 51 ZP

>C 51
```

②ワープロで作った ASCII ファイルをルーチンとして登録する

```
>O 51:("A:XFER.RTN":"R")

>U 51 ZL

>C 51

>ZS ABC
```

6 他のシステムの呼び出し

SP-MUMPS には，$ZCALL 関数という拡張機能が用意されています。この関数を利用すると，MUMPS を起動したままで他のシステムを呼び出すことができます。

ただし，これには前提条件があります。

①コンピュータの内部記憶装置が512KB以上あること

②MUMPS を起動したディスク装置に MS-DOS の COMMAND.COM のファイルがあること

です。この条件が満たされた状態で以下の通りの処理ができます。

たとえば

```
>S X=$ZCALL("COMMAND.COM")↵
```

とすると

```
COMMAND.COM ﾊﾞｰｼﾞｮﾝ 2.11

A>
```

と表示されます。これは MS-DOS の起動された状態です。ここで MS-DOS の命令を実行することができます。

```
A>DIR↵
```

MS-DOS から MUMPS へ戻るには

```
A>EXIT↵
```

とします。するとすぐにまた MUMPS に戻ります。

```
>                    :MUMPSのプロンプトです
```

さらに

```
>S X=$ZCALL("COMMAND.COM /CDIR")↵
```

とすると，MS-DOS の DIR コマンドを直接実行することができます。この命令をルーチンとしてセットしておくと便利です。

次に DIR の結果の例を示します。

```
>S X=$ZCALL("COMMAND.COM /CDIR A:")↵

 ドライブ A: のディスクにはボリュームラベルがありません
 ディレクトリは A:¥

MUMPS      EXE    223124   86-01-23   19:36
MSM        COM      1241   85-12-13   15:02
MSMZCALL   COM      1743   85-12-13   17:02
UTILS      MGR    129451   86-01-21   14:08
GLOBALS    MGR     52785   86-01-18   19:43
INSTALL    DB      81920   86-01-14   16:57
%RELOAD    UTL       417   85-12-23   12:59
PATCH      MGR      3436   86-01-31   11:33
        8 個のファイルがあります
   150528 バイトが使用可能です

>
```

こうして，MUMPS を終了することなく MS-DOS を呼び出せるので，ディスク上の MUMPS 以外のファイルの内容を調べたりすることや，さらにMS-DOS に乗っている他のシステムとデータを交換することができます。

日本語ワープロの「一太郎」で MUMPS のルーチンを編集するためのルーチンを例としてあげておきます。

list 3-6

```
%UE     ;YS,UTILITY;1986.2.11;一太郎どのリンク
        ;
1       K ^%UE
        S ^%UE=$P($T(TXT),"  ",2,9999)
        F J=1:1 S T=$T(TXT+J) Q:T=""  S ^%UE($P(T,"  ",1))=$P(T,"  ",2,9999)
        Q
TXT     S %="INIT" F %QQ=0:0 X ^%UE(%) I %="END" X ^%UE("KILL") Q
INIT    S %="END" Q:$T(+1)=""  S %="CHDIR" Q
CHDIR   S %="FILNAM",%X=$ZCALL("COMMAND.COM /CCD ¥JXW") Q
FILNAM  S %="SAVE",%FILNAM="%UE.TXT" Q
SAVE    S %="JXW" O 51:(%FILNAM:"W") U 51 ZP  W ! C 51 Q
JXW     S %="LOAD",%X=$ZCALL("JXW.COM "_%FILNAM) Q
LOAD    S %="END" O 51:(%FILNAM:"R") U 51 ZL  C 51 Q
KILL    K %,%QQ,%X Q
```

このルーチンは，^%UEというグローバル変数をセットします。そしてそのグローバル変数を XECUTE 命令で呼び出すと，そのときワークエリアにあるルーチンを「一太郎」に渡し編集することができます。

```
>D ^%UE↵        ：グローバル変数 ^%UE のセット
```

と，1回このルーチンを実行しておくと，その後は

```
>X ^%UE↵        :「一太郎」の呼び出し
```

で，「一太郎」を呼び出すことができます。

「一太郎」とのデータのやり取りに，%UE.TXT というファイルを用いています。また，「一太郎」はカレントディスク装置のディレクトリ¥JXW¥にあるものと想定しています。これらの設定が不都合な場合は，CD ¥JXW の部分を変更してください。

使用上の注意としては，ワープロで作成したテキストが，MUMPS のルーチンとして正しい形式になっていなくてはならないことです。

文字のない行や，行ラベルに許されない文字があると，MUMPS に戻った時点で〈ISYTX〉というエラーになってしまいます。とくに，ワープロで扱う空白文字には「全角」の空白と，1バイト文字（しばしば「半角」文字ともいう）の空白があります。これらは見た目には幅が違うだけですが，MUMPS にとってはまったく異なった文字です。そして命令の区切りなどに「全角」の空白は使用できないので注意してください。

また「一太郎」を終了する際に，%UE.TXTというファイルに編集を終わったルーチンを忘れず格納してください。さもないと，せっかく編集したルーチンが消えてしまいます。

ルーチンの編集に，ワープロやテキストエディタを用いる利点は，漢字入力が容易でカーソルを自由に移動して文字が編集でき，テキストのコピーや移動が容易だということです。

しかし MUMPS が呼び出す相手が大きいと，呼び出しに時間がかかります。

このような方法を実用的なものにするには，ハードディスク装置や RAM ディスク装置があったほうがよいでしょう。

file 4 ユーティリティプログラム

プログラムの開発や保守に「道具」として使用するプログラムをユーティリティプログラムと呼んでいます。MUMPS では MUMPS 自身を用いてユーティリティを作ることができます。ここではルーチンをリストするユーティリティとルーチンをディスクに格納するユーティリティを紹介します。

1 ルーチンのリスト

ルーチンのリストをプリンタに印刷するもっとも簡単な方法は

```
>O 3 U 3 ZP↵
```

これで，ワークエリア上にあったルーチンがプリンタに出力されます。

```
O 3
```

は，OPEN 命令で，プリンタへデータを渡す扉（3番の扉）を開くという命令です。

```
U 3
```

は USE 命令で，3番の扉を用いてデータを渡すという命令です。

そして，ZPRINT 命令でルーチンをプリンタに渡します。

この方法は簡単ですが原始的です。

リスト4-1に掲げるルーチンは，ルーチンをプリントするためのルーチン（プログラムリスタ）です。ルーチンの命令行の他に，見出し，印刷年月日と時刻，行番号などを印刷してくれます。

プログラムリスタの使用例として，プログラムリスタ自身を出力します。

まず，プログラムリスタを呼び出します。

```
>D ^%UPL↵
```

するとタイトルが表示され，印刷したいルーチンの名前をたずねてきます。

```
                    *** プログラム・リスタ ***
ルーチンの選択:
```

ここで出力するルーチンを選びます。

```
ルーチンの選択: %UPL↵      :プログラムリスタ自身を選びます

        1 ルーチンが選択されました

ルーチンの選択:
```

するとまたルーチン名をたずねてきます。

ここでは%UPLだけを出力したいので↵キーのみを打ちます。もし，複数のルーチンを続けてリストしたければここで名前を入力してください。『？』記号を入れると，ルーチン名の指定方法の説明が表示されます。

```
ルーチンの選択: ↵      :↵キーのみ打ちます
```

次に出力装置を選びます。プリンタは3番です。↵キーのみで3が選ばれたことになります。もし，モニターテレビに表示したければ0番を，ディスクにASCII ファイルとして出力したければ51番を指定してください。

```
出力装置を入力して下さい <3>: ↵   :↵キーのみなら装置3（プリンタ）
                                    を選ぶことになります
```

さらに出力のページの大きさ，印刷するタイトル，部数についてたずねてきます。

「〈 〉」で囲まれた値で良ければ，↵キーのみを打ってください。もし変更したければその数値を入力してください。

```
1 ページの行数 <60>:47↵
1 行の最大文字数 <132>: 70↵
任意の見出し: プログラム・リスタの出力例↵
* リストは何部必要ですか ? <1>: ↵ ==> 1
```

最後にこの設定でよいかどうかたずねてきます。YかNを入力してください。

```
* リストの準備はよろしいですか ? <はい>: ↵ ==> はい
```

ここで，ルーチンがプリンタに出力されるはずです。

以上の手順で，『＾』記号を入力すると，すぐ前の質問に戻ります。間違えたときに利用してください。

出力が終わると，またルーチンの名前をたずねてきます。もしまだ出力すべきルーチンがあれば繰り返して指定できます。↵キーのみを入力すると，プログラムリスタは終了します。

```
ルーチンの選択: ↵
     !!! ルーチンが選択されませんでした !!!

*** プログラム・リスタ  終了 ***
```

リスト4-1に出力結果を掲げます。

list 4-1

```
プログラム・リスタの出力例                1986.2.18 12:18      [%UPL]   Page 1

 1   :%UPL      ;YS,,UTILITY,SP-MUMPS;1986/2/18 11:56;ﾌﾟﾛｸﾞﾗﾑ ﾘｽﾀ
 2 +1:          ;
 3   :START     U 0 W *26,?25,"*** プログラム・リスタ ***",!!
 4 +1:          S %U="^",%ZW=4
 5 +2:          S %CTRL=1 F %QQ=0:0 D DRIVE I %CTRL=0 Q
 6   :END       K:'$D(DEBUG) %Q,%QQ,%P,%HDR,%C,%CC,%CTRL,%DEV,%LBL,%OFFSET
          -->,%SIZ,%I,%S,%J,%RTN,%X1,%X2,%X3,%X4,%X5,%X6,%X7,%LIN,%LPP
 7 +1:          U 0 W !!,"*** プログラム・リスタ  終了 ***",!
 8 +2:          QUIT
 9   :DRIVE     I %CTRL=1 D SELRTN S %CTRL=$S(%CTRL=0:0,1:2) Q
10 +1:          I %CTRL=2 D SELDEV S %CTRL=$S(%DEV=%U:1,1:3) Q
11 +2:          I %CTRL=3 D RLPP S %CTRL=$S(%LPP=%U:2,1:4) Q
12 +3:          I %CTRL=4 D RWDTH S %CTRL=$S(%WDTH=%U:3,1:5) Q
13 +4:          I %CTRL=5 D SETIOF S %CTRL=$S(%IOF=%U:4,1:6)
14 +5:          I %CTRL=6 D RHDR S %CTRL=$S(%HDR=%U:4,1:7) Q
15 +6:          I %CTRL=7 D RNOCPY S %CTRL=$S(%NOCPY=%U:6,1:8) Q
16 +7:          I %CTRL=8 D ROK S %CTRL=$S(%OK=%U:7,1:9) Q
17 +8:          I %CTRL=9 D LIST S %CTRL=1
18 +9:          Q
19   :SELRTN    S %DEV=$I U 0 D ^%RSEL
20 +1:          I '$D(^UTILITY($J)) W !?5,"!!! ルーチンが選択されませんで
          -->した !!!" S %CTRL=0
21 +2:          Q
22   :SELDEV    D PR^%SDEV
23 +1:          I $D(QUIT) S %DEV=%U
24 +2:          Q
25   :RLPP      S %LPP=60 I $D(^%UDEV(%DEV,"LPP")) S %LPP=^%UDEV(%DEV,"LPP
          -->")
26 +1:          S %LOOP=1 F %QQ=0:0 D RLPP1 Q:%LOOP=0
27 +2:          I %LPP=+%LPP,%LPP>0 S ^%UDEV(%DEV,"LPP")=%LPP
28 +3:          Q
29   :RLPP1     W !!,"1 ページ中の行数 <",%LPP,">: " R %X
30 +1:          I %X="" S %LOOP=0 W "==>",%LPP Q
31 +2:          I %X=%U S %LPP=%U,%LOOP=0 Q
32 +3:          I %X'?1N.N!(%X'=+%X)!(%X<5) W !?5,"??? 5 以上の整数を入れ
          -->て下さい ???" S %LOOP=1 Q
33 +4:          S %LOOP=0,%LPP=%X
34 +5:          Q
35   :RWDTH     S %WDTH=132 I $D(^%UDEV(%DEV,"CPL")) S %WDTH=^%UDEV(%DEV,"
          -->CPL")
```

```
36 +1:          S %LOOP=1 F %QQ=0:0 D RWDTH1 Q:%LOOP=0
37 +2:          I %WDTH=+%WDTH,%WDTH>0 S ^%UDEV(%DEV,"CPL")=%WDTH
38 +3:          Q
39   :RWDTH1    W !,"1 行中の最大文字数 <",%WDTH,">: " R %X
40 +1:          I %X="" S %LOOP=0 W "==>",%WDTH Q
41 +2:          I %X=%U S %WDTH=%U,%LOOP=0 Q
42 +3:          I %X'?1N.N!(%X'=+%X)!(%X<10)!(%X>132) W !?5,"??? 10 から 1
          -->32 までの整数を入れて下さい ???" S %LOOP=1 Q
43 +4:          S %LOOP=0,%WDTH=%X
44 +5:          Q
45   :SETIOF    I $D(^%UDEV(%DEV,"IOF")) S %IOF=^%UDEV(%DEV,"IOF") Q
46 +1:          S %LOOP=1 F %QQ=0:0 D SETIOF1 Q:%LOOP=0
47 +2:          I %U'[%IOF S ^%UDEV(%DEV,"IOF")=%IOF
48 +3:          Q
49   :SETIOF1   R !,"改頁の手順 (WRITE命令の引数) ? <#>:",%IOF
50 +1:          I %IOF="" S %IOF="#" W "==> #" S %LOOP=0 Q
51 +2:          I %IOF?1"?".E W !?5,"!!! 改頁のためのWRITE命令の引数を入力
          -->してください !!!",!?10,"例: モニターテレビ  ==>  *26,#",!
          --> S %LOOP=1 Q
52 +3:          S %LOOP=0
53 +4:          Q
54   :RHDR      S %MXLEN=%WDTH-38
55 +1:          S %LOOP=1 F %QQ=0:0 D RHDR1 Q:%LOOP=0
56 +2:          Q
57   :RHDR1     R !,"任意の見出し: ",%HDR
58 +1:          I %HDR?1"?".E W !?5,"!!! 自由文章を入れて下さい !!!" S %LO
          -->OP=1 Q
59 +2:          I $ZW(%HDR,%ZW)>%MXLEN W !?5,"??? 長過ぎます ??? 半角文字
          -->で ",%MXLEN," 文字以内にしてください ???" S %LOOP=1 Q
60 +3:          S %LOOP=0
61 +4:          Q
62   :RNOCPY    S %LOOP=1 F %QQ=0:0 D RNOCPY1 Q:%LOOP=0
63 +1:          Q
64   :RNOCPY1   R !,"* リストは何部必要ですか ? <1>:",%NOCPY
65 +1:          I %NOCPY=%U S %LOOP=0 Q
66 +2:          I %NOCPY="" S %NOCPY=1,%LOOP=0 W "==> 1" Q
67 +3:          I %NOCPY'=+%NOCPY W !?5,"??? 整数を入れてください ???" S %
          -->LOOP=1 Q
68 +4:          S %LOOP=0
69 +5:          Q
70   :ROK       S %LOOP=1 F %QQ=0:0 D ROK1 Q:%LOOP=0
71 +1:          Q
72   :ROK1      R !,"* リストの準備はよろしいですか ? <はい>: ",%OK
73 +1:          I %OK=""!(%OK="Y")!(%OK="y")!(%OK="ﾊ")!(%OK="ﾝ") W "==> は
          -->い" S %OK="Y",%LOOP=0 Q
74 +2:          I %OK=%U!(%OK="N")!(%OK="n")!(%OK="ｲ")!(%OK="ﾐ") W "==> い
          -->いえ" S %OK=%U,%LOOP=0 Q
75 +3:          W !?5,"??? Y(=はい) または N(=いいえ) を入力してください ?
          -->??"
76 +4:          S %LOOP=1
77 +5:          Q
78   :LIST      F %I=1:1 S %X=$T(TXT+%I) Q:%X=""!(%X=" ;***")  S @$P(%X,"
          -->",1)=$P(%X," ",2,999)
```

```
 79 +1:          S %LIM=%WDTH-16 D DATTIM
 80 +2:          U %DEV
 81 +3:          F %NNN=1:1:%NOCPY S %RTN="" X %X1
 82 +4:          I %DEV>1 C %DEV
 83 +5:          Q
 84   :DATTIM    S %H=$H D TIME,DATE
 85 +1:          Q
 86   :TIME      S %TIME=$P(%H,",",2),%A=%TIME#60,%TIME=%A#60_":"_$E(%A#60+
               -->100,2,3)
 87 +1:          Q
 88   :DATE      I %H<21609 S %H=%H-1
 89 +1:          S %A=%H#1461,%A=%A>58+(%A>423)+(%A>788)+%A
 90 +2:          S %YY=%H#1461*4+1841+(%A#366),%B=%A#366
 91 +3:          S %C="31/60/91/121/152/182/213/244/274/305/335/366"
 92 +4:          F %MM=1:1:12 I %B<$P(%C,"/",%MM) Q
 93 +5:          S %DD=%B-$P(%C,"/",%MM-1)+1
 94 +6:          S %DATE=%YY_"."_%MM_"."_%DD
 95 +7:          Q
 96   :TXT       ;XECUTE命令のテキスト
 97   :%X1       F %I=1:1 S %RTN=$O(^UTILITY($J,%RTN)) Q:%RTN=""  ZL @%RTN
               -->S %P=0 X %X2 W @%IOF
 98   :%X2       S %OFFSET=0 F %J=1:1 X:%J=1!($Y>%LPP) %X4 S %X=$T(+%J) Q:%
               -->X=""  X %X3:0,%X7 S %X=$P(%X," ",2,999) F %K=1:1 X %X5 S %
               -->Y=$E(%X,1,%Q),%X=$E(%X,%Q+1,999) Q:%Y=""  W ?16,%Y W:%X'="
               -->" !?13,"-->"
 99   :%X3       I %COMT S %C=$P(%X," ",1) I %C'="",$D(^%C(%RTN,%C,0)) S %C
               -->=0 W ! F %CC=1:1 S %C=$O(^(%C)) W:%C="" ! Q:%C=""  X:$Y>%L
               -->PP %X4 W !,?10,";; ",^(%C)
100   :%X4       W:%P @%IOF S %P=%P+1 W !,%HDR,?%WDTH-38,%DATE," ",%TIME,?%
               -->WDTH-13-$L(%RTN),"[",%RTN,"]",?%WDTH-8,"Page ",%P,!
101   :%X5       S %Q=$L(%X),%Y=%X I $ZW(%X,%ZW)>%LIM X %X6 F %Q=%Q:-1:%Q-6
               --> S %Y=$E(%X,1,%Q) I $ZW(%Y,%ZW)'>%LIM Q
102   :%X6       F %Q=%LIM#2:7:%Q+6 S %Y=$E(%X,1,%Q) I $ZW(%Y,%ZW)'<%LIM Q
103   :%X7       S %LBL=$P(%X," ",1),%OFFSET=$S(%LBL="":%OFFSET+1,1:0) W !,
               -->$J(%J,3) W:%OFFSET $J("+"_%OFFSET,3) W ?6,":",%LBL," "
104 +1:          ;***
```

②%USユーティリティ

ルーチンをその第1行目のラベル名でディスクに登録するプログラムを紹介します。リスト4-2に掲げるルーチンを1回実行してください。それ以降

```
>X ^%US↵
```

で，ルーチンを自動的に登録します。その際，ルーチンヘッダーの日付と時刻を更新してくれます。登録すると

```
[####] はディスクに格納されました。
```

と表示されます。

list 4-2

```
%US     ;YS,UTILITY;1986.2.18 13:02;SET ^%US
        ;
0       U 0 W !,"LOAD ^%US"
        K ^%US
1       S ^%US=$P($T(TXT)," ",2,9999)
        F I=1:1 S T=$T(TXT+I) Q:T=""!(T=" ;***")  S ^%US($P(T," ",1))=$P(T," ",2
     -->,9999)
        W !,"LOAD END",!!
        K I,T
        QUIT
TXT     S %T=$T(+1) Q:%T=""  X:$P(%T," ",2)?1";".E ^%US("%DATE"),^%US("%DATE1"),
     -->^%US("%DATTIM"),^%US("%HEADER") X ^%US("SAVE") K %RTN,%T,%DATE,%DT,%TIME
     -->,%X,%Y,%H,%L,%YR,%M,%D,%I
%DATE   S %DT=$P($H,",",1) Q:%DT<0  S %H=%DT-(%DT<21609),%L=%H¥1461,%YR=%H#1461,
     -->%Y=%L*4+1841+(%YR¥365),%D=%YR#365+1,%M=1 S:%YR=1460 %D=366,%Y=%Y-1
%DATE1  F %I=31,%Y#4=0+28,31,30,31,30,31,31,30,31,30,31 Q:%D'>%I  S %D=%D-%I,%M=
     -->%M+1
%DATTIM S %DATE=%Y_"."_%M_"."_%D,%TIME=$P($H,",",2)¥60,%TIME=%TIME¥60_":"_$E(%TI
     -->ME#60+100,2,3)
%HEADER S %T=$P(%T,";",1,2)_";"_%DATE_" "_%TIME_";"_$P(%T,";",4,9999) ZR +1 ZI %
     -->T
SAVE    S %RTN=$P(%T," ",1) ZS @%RTN W !,"[",%RTN,"] はディスクに格納されました。
     -->"
        ;***
```

このユーティリティは，ルーチンの第1行目が Chapter 3-12 で解説している「ルーチンヘッダーの慣例」にしたがっていることを前提にしています。注意してください。

file 5 MUMPSの文法要約

1 データ

MUMPS の扱うデータはすべて可変長文字列です。ANSI の標準 MUMPS ではひとつのデータの最大長は 255 文字です。必要に応じてこれらを数値計算に用いることもできます。数値計算の精度は標準 MUMPS では 9 桁までが保障されています。

```
ABC
12.567
-100E10
```

プログラム上に直接データの値を記述したものを定字型といい，数値はそのまま記載できるが，それ以外の文字列は引用符『"』で囲う必要があります。

```
定字列 ┌ 定数字列  12345  -93.5455  9.87E-2
       └ 定文字列  "ABC"  "56kg"
```

文字のないデータを空文字といい，プログラム上『""』で表わされます。

2 変数

ローカル変数は，普通コンピュータ内部のワークエリア上に記憶され，一時的なデータの記憶に利用され，グローバル変数は通常ディスク上に記憶されるので半永久的なデータベースとして利用されます。いずれの変数も添え字を任意に数個もつことができ，添え字には文字列データが使えます，また，値が連続している必要はありません。あらかじめデータ数や，添え字の値や個数を宣言する必要もありません。

ローカル変数とグローバル変数は先頭に『^』が付いているか否かで区別されます。

付いているのがグローバル変数，付いていないのがローカル変数です。

ローカル変数の例：　X, ABC, %A, SYS1, DATA(59), VAR(1,5,999)
グローバル変数の例：　^X, ^ABC, ^%, ^SYS, ^PT(12345,"ABC",3)

特殊変数は MUMPS システムが自動的にセットし，これを利用すると，ユーザーはシステムの状態を調べることができます。標準 MUMPS の特殊変数は次の通りです。

名前	内容
$HOROLOG	日付と時間　d,t d は1840年12月31日を0とした日数 t はその日の0:00am からの秒数
$IO	そのとき使用している入出力装置番号
$JOB	MUMPSシステム全体のプロセス番号
$STORAGE	ワークエリアの残りスペースの大きさ
$TEST	IF命令と時間制限付き命令の実行結果のフラグ
$X	1行に出力した文字数のカウンタ
$Y	1頁に出力した行数のカウンタ

標準 MUMPS より拡張された特殊変数の名前には**Z**を付けることになっています。たとえば，$ZA, $ZERROR, $ZVERSION などです。

3 関数

関数は，(　) 内に指定された情報をもとに，データを作り出す機能です。

標準 MUMPS より拡張された関数の名前にはすべて Z を付けることになっており，たとえば $ZEDIT, $ZFUNC などです。

名前	機能	例	値
$ASCII	文字の内部コード	$A("A") $A("AB",2)	65 66
$CHAR	指定されたコードに対応する文字	$C(65,66,67)	ABC
$DATA	変数の定義状態	$D(^X(1))	^X(1) が存在しなければ0 ^X(1) が存在するが、下位節がなければ1 ^X(1) は存在しないが、下位節があれば10 ^X(1) が存在し、下位節もあれば11

$EXTRACT	データの抽出	$E("ABC") $E("ABC",2) $E("ABC",1,2)	A B AB
$FIND	あるデータの中に含まれる部分データの位置	$F("ABCA","A") $F("ABCA","A",2) $F("ABCA","D") $F("ABCA","")	2 5 0 1
$JUSTIFY	データを右詰めにする データを四捨五入する	$J("ABC",5) $J(1.234,7,2)	__ABC ___1.23
$LENGTH	データに含まれる文字数	$L("") $L("ABC") $L("ABAC","A")	0 3 3
$ORDER	指定された変数の兄弟節の中のすぐ上の兄の添え字	$O(A(3))	A(4) があれば 4
$PIECE	データをある境界子をもとに分解したデータ	$P("ABC","B",1) $P("ABC","B",2) $P("ABC","B") $P("ABCABD","B",2,3)	A C A CABD
$RANDOM	乱数	$R(100) $R(2)	0 から 99 までの乱数 0 か 1
$SELECT	場合分け	$S(X=1:"A",X=2:"B",1:"Z")	X=1 なら A X=2 なら B それ以外なら Z
$TEXT	ルーチンの行内容	$T(+1) $T(ABC)	そのときワークエリア上にあるルーチンの第1行目 ワークエリア上のルーチンの行 ABC の内容
$VIEW	装備により異なる		

4 演算子

演算子はひとつ以上のデータから新たなデータを算出する機能です。

以下の演算子のうち，符号の＋，－と論理否定の ' は単項演算子で，演算子の右側に演算対象があります。

それ以外の演算子はみな二項演算子で，両側に演算対象があります。

演算子	例	解説
正符号： +	+10.5 +"20cm"	10.5 20　文字列を数値解釈する機能がある
負符号： -	-10.5 -"98kg"	-10.5 -98
加算： +	1+5 A+7	6 変数 A の値に 7 を加えた値
減算： -	3-9	-6
乗算： *	4*5.3	21.2
除算： /	9/5	1.8
整数除算：¥（または＼）	9/5	1　除算の整数部分
モジュロ（法）： #	9#5	4　除算の余り
大なり： >	5>3 X>0	1　数値の大小を比較する条件式 X が 0 より大きければ 1 そうでなければ 0
小なり： <	5<3 X<0	0 X が 0 より小さければ 1 そうでなければ 0
等しい： =	DATA="ABC" 0.5="0.5"	DATA の値が ABC なら 1 そうでなければ 0 0　これは文字列データが等しいか否かの条件式で、 0.5 は定数で .5 と解釈される。.5 は 0.5 とは文字として等しくないので 0
包含： [	"ABC"["C"	1　左辺のデータが右辺のデータを含む 判断する条件式に用いる
追従：]	"AAA"]"AB"	0　文字の内部コードの大小を比較する 辞書配列と考えると分かりやすい
連結： _ （下線）	"I love "_X	X="you" なら I love you X="her" なら I love her 2 つのデータ文字列を連結する
パターン照合： ?	"ABC"?3A "10"?2.N "10.5"?1.5N "10.5"?2N1P1N "10/5"?.2N1"/"1.N	右辺のデータが左辺のパターンに合うか否かを判断する 1 1 0 1 1 （後記「パターン子」参照）

論理積： & （および）	(X="A")&(Y="B")	X="A" かつ Y="B" なら 1 両辺の条件式の論理積
論理和： ! （または）	(X=5)!(Y=10)	X=5 または Y=10 なら 1 両辺の条件式の論理和
論理否定： ' （アポストロフィ）	'0 '1	1 0 否定演算子は、条件判断に用いる他の演算子と組み合わせて、否定の条件をつくることができる 例： '= 等しくない '> 大きくない '] 含んでいない

パターン

パターンはいくつかのパターン子からなっています。ひとつのパターン子は，乗数，パターンコードまたは定文字列からなっています。

パターン・コード	意味
A	アルファベット
U	アルファベットの大文字
L	アルファベットの小文字
P	記号
N	数字
C	制御文字
E	すべての文字
カ	すべてのカナ
オ	カナの大文字
ヲ	カナの小文字
ト	カナの濁点，半濁点，長音
全	2バイト文字（漢字など[11]）
"ABC"	パターンコードの代わり，文字そのものを表す

乗数

整数	その整数の値
（ピリオド）	いくつでもよい（0も含む）
.整数	整数以下
整数.	整数以上
整数1 . 整数2	整数1以上，整数2以下

■notes

10) これは日本語MUMPS標準化委員会の仕様。SP-MUMPSでは「T」となっている。
ただし，近い将来SP-MUMPSも標準化委員会案に従うはずである

5 式

式は，変数，定数，関数などの演算対象と，演算子の組み合わせです。

演算子はすべて，ひとつの式の中で混合して用いることができますが，その場合単項演算子は二項演算子に優先します。単項演算子同士，二項演算子同士では優先順位がありません。ただし括弧（ ）があるときは，それが優先され括弧がいくつもあるときには，もっとも内側のものが優先されます。

```
      結果                                  式
1+2/5     ---> .6
"2dogs"+"3cats"_"pigs"  ---> 5pigs
$E(MM+100,2,3)_"日"     ---> MM=5 なら 05日
(NAME["ｼ")!(SEX="F")  ---> NAME が ｼ を含むかまたは SEX が F なら 1
                           そうでなければ 0
```

6 命令

命令は，MUMPS が実行すべき処理内容を指示するものです。命令はその命令に特有な名前と，処理に必要な引数からなっています。

名前	機能	例	解説
BREAK	ルーチンを一時中止する	B	
CLOSE	装置の占有権を放棄する チャネルを閉じる	C 3	番号 3 の装置のチャンネルを閉じる
DO	モジュールの呼び出し	D ABC	ワークエリア上の行 ABC 以降を呼び出す
		D ^XYZ	ディスク上のルーチン XYZ を呼び出す
		D LINE:X=2	X=2 であれば行 LINE を呼び出す
ELSE	$TEST 特殊変数による 条件付け	E S X=5	$T=1 なら S X=5 を実行する
FOR	命令の繰り返しの制御	F J=1:1:10	右にある命令を10回繰り返す
		F QQ=0:0	無限循環
		F SEX="M","F"	SEX の値を M と F にセットしてそのたびに右にある命令を繰り返す 結局 2 回繰り返す

GOTO	指定された行またはルーチンへジャンプ	G ABC G ^XYZ	行 ABC へ行く ルーチン XYZ へ行く
HANG	指定秒数処理を停止	H 10	10 秒間停止
HALT	ＭＵＭＰＳを終了	H H:X=""	ＭＵＭＰＳを終了 X="" なら終了
IF	右にある命令を条件付ける	I N>0 D LIST I W "OK"	N>0 なら行 LIST を呼び出す 条件が真なら $TEST を 1 にセット。偽なら0にセット $TEST 特殊変数の値が 1 なら OK と表示
JOB	他のＭＵＭＰＳプロセスを起動する	J ABC J ^ABC	行 ABC 以降を別プロセスとして呼び出す ルーチン ABC を別プロセスとして呼び出す
KILL	変数を削除する	K X,Y,Z(5),^Q K (A) K	変数 X,Y,Z(5),^Q と、その下位節を削除する ローカル変数 A とその下位節以外のローカル変数を削除する すべてのローカル変数を削除する
LOCK	マルチジョブの調整	L A L L (A,B)	他のプロセスが変数 A を LOCK することを禁止する LOCK していた変数を解放 A と B を同時に LOCK する
NEW	変数のスタック	N A,B,C N (X) N	FOR と無関係な QUIT 命令が実行されると元の変数に戻る
OPEN	装置の占有権を確保する 装置のチャネルを開く	O 3 O 51:("ABC":"R") O 4::10	装置3の占有権を確保 装置51の占有権を確保し、チャネルを開く。その際 "ABC":"R"というパラメータを用いる 10秒以内に装置4の占有権が得られないと、$TEST 特殊変数を 0 にセットしてこの命令を終了する
QUIT	同一行の左側に FOR 命令があればそれを終了させるなければ、そのモジュールを呼び出した DO 命令または XECUTE 命令の次へ戻る	Q Q:X=""	FOR 命令、DO 命令、XECUTE 命令のいずれかを終了させる X="" なら QUIT する
READ	データを読み込む	R X	データを読み込み、変数 X にセットする

		R !,"DATA ? ",A	改行し、 DATA ? と表示し、データを読み込み、それを変数 A にセットする
SET	変数に値を割り付ける	S X=1	変数 X にデータ 1 を割り付ける
		S ^PI(ID,ITEM)=DATA	ID=5, ITEM="RBC", DATA="450" のとき、変数 ^PI(5,"RBC") に 450 というデータを格納する
		S (A,B)=0	変数 A と B に 0 をセット
		S $P(DATE,"/",2)=MM	変数 DATE の値を / で区切ったとき、その2番目のところに MM の値を代入する DATE="1986/2/21", MM="5" なら S DATE="1986/5/21" と等価である
USE	以降の入出力をする装置を宣言する	U 0	以降の入出力は、ログインした端末で行う
		U 3	以降の入出力は装置番号3で行う $IO 特殊変数の値は 3 になる
VIEW	装備ごとに異なる		
WRITE	出力する	W "ABC"	ABC と表示する
		W X+2	X+2 の値を表示する
		W !,"DATA=",X(N)	改行し、DATA= と表示し、さらに X(N) の値を表示する
XECUTE	データを実行する	X "S A=5"	S A=5 と等しい
		X ^%US	グローバル変数 ^%US の値を命令として実行する

標準 MUMPS より拡張された命令の頭文字には Z を付けることになっており，たとえば，ZREMOVE 命令，ZFLUSH 命令などです。

WRITE／READ命令の補足

W＊Z　内部コードZの文字を1文字出力する
R ＊X　データを1文字読み込む。変数Xにはその文字の内部コードがセットされる
R X:#5　データを5文字読み込む
R X:8　データ入力完了まで最大8秒待つ。8秒以内に完了するとST特殊変数の値が1になり，完了せずに次の命令へ移るとSTは0にセットされる

7 ルーチン

MUMPS のプログラムはルーチンからなり，ルーチンはいくつかの行からなってます。ひとつの行は

行ラベル TAB 命令　　または　　TAB 命令

という形をもち，ラベルはその行に付けた名前です。

A，B，　X00，　0012，%A，　%1Z

という形をしており，ラベルはその行に付けた名前です。

ルーチン行の命令の代わりに，；記号を入れると，その右側は注釈文と解釈され，命令として実行されません。

8 間接指定

間接指定は，データを命令またはその一部として実行する機能です。

XECUTE 命令も間接指定のひとつです（file 5-6 参照）。

①パターンレベル

パターン照合演算子の右にあるパターンを間接指定で置き換えができます。

```
X?@P      P="1.A"   ならば    X?1.A
          P="2UN"   ならば    X?2UN
```

②名前レベル

変数名，行ラベル，ルーチン名を間接指定で置き換えることができます。

```
S @V=1    V="A"       ならば    S A=1
          V="^X(65)"  ならば    S ^X(65)=1

D ^@X     X="ABC"     ならば    D ^ABC
          X="QQ"      ならば    D ^QQ

D @X^ABC  X="ABC"     ならば    D ABC^ABC
          X="QQ"      ならば    D QQ^ABC
```

③引数レベル

FOR 命令と，IF 命令以外の，引数を持つ命令は，引数を間接指定で置き換えることができます。

```
S @V      V="A=5"         ならば    S A=5
          V="P=3,R=10"    ならば    S P=3,R=10

D @E      E="ABC"             ならば    D ABC
          E="^PQR"            ならば    D ^PQR
          E="ABC,XXX,^RRR"    ならば    D ABC,XXX,^RRR
```

file 6 JIS (ASCII) コード表

コンピュータ内部では，文字はすべて対応する数値として記憶されています。この表は，その文字と数値の対応を示しています。

JIS 規格（C-6220）に準拠していますが，コード0から127までは，米国国家規格協会（ANSI）が定めている ASCII コードに一致しています。JIS と ASCII で唯一異なるのは，JIS で92番は『¥』記号であるのに対して，ASCII では92番が『＼』記号である点です。このため，MUMPS の整数除算の演算子は装置によって，『¥』になったり『＼』になったりします。ただしコンピュータにとっては，内部的にはどちらも数値コード92番なので，同じ文字として認識されます。

コード0から31の文字と127の文字は制御文字で，表示しても見えません。ときに正しいはずのルーチンが「〈SYNTX〉エラー」を起こすことがありますが，これらの制御文字が紛れ込んでいるかもしれません。いったんその行を消して，再入力してください。

コードの対応を確かめるには，$ASCII 関数と，$CHAR 関数を用います。W $A(X) で文字Xの内部コードが，W $(Y) で内部コードがYである文字が表示されます。一連の対応を表示するには次の通りの命令を実行してください。

コード	文字	コード	文字	コード	文字	コード	文字	コード	文字	コード	文字	コード	文字	コード	文字
0	NUL	32	(空白)	64	@	96	`	128	†	160	(空白)	192	タ	224	†
1	SOH	33	!	65	A	97	a	129	¡	161	。	193	チ	225	¡
2	STX	34	"	66	B	98	b	130	¡	162	「	194	ツ	226	¡
3	ETX	35	#	67	C	99	c	131	¡	163	」	195	テ	227	¡
4	EOT	36	$	68	D	100	d	132	¡	164	、	196	ト	228	¡
5	ENQ	37	%	69	E	101	e	133	¡	165	・	197	ナ	229	¡
6	ACK	38	&	70	F	102	f	134	¡	166	ヲ	198	ニ	230	¡
7	BEL	39	'	71	G	103	g	135	¡	167	ァ	199	ヌ	231	¡
8	BS	40	(	72	H	104	h	136	¡	168	ィ	200	ネ	232	¡
9	HT	41	)	73	I	105	i	137	未	169	ゥ	201	ノ	233	未
10	LF	42	*	74	J	106	j	138		170	ェ	202	ハ	234	
11	VT	43	+	75	K	107	k	139	定	171	ォ	203	ヒ	235	定
12	FF	44	,	76	L	108	l	140		172	ャ	204	フ	236	
13	CR	45	-	77	M	109	m	141		173	ュ	205	ヘ	237	
14	SO	46	.	78	N	110	n	142	義	174	ョ	206	ホ	238	義

15	SI	47	/	79	O	111	o	143	未	175	ｯ	207	ﾏ	239	未
16	DLE	48	0	80	P	112	p	144		176	ｰ	208	ﾐ	240	
17	DC1	49	1	81	Q	113	q	145	定	177	ｱ	209	ﾑ	241	定
18	DC2	50	2	82	R	114	r	146		178	ｲ	210	ﾒ	242	
19	DC3	51	3	83	S	115	s	147		179	ｳ	211	ﾓ	243	
20	DC4	52	4	84	T	116	t	148	義	180	ｴ	212	ﾔ	244	義
21	NAK	53	5	85	U	117	u	149		181	ｵ	213	ｭ	245	
22	SYN	54	6	86	V	118	v	150		182	ｶ	214	ﾕ	246	
23	ETB	55	7	87	W	119	w	151		183	ｷ	215	ﾗ	247	
24	CAN	56	8	88	X	120	x	152		184	ｸ	216	ﾘ	248	
25	EM	57	9	89	Y	121	y	153		185	ｹ	217	ﾙ	249	
26	SUB	58	:	90	Z	122	z	154		186	ｺ	218	ﾚ	250	
27	ESC	59	;	91	[	123	{	155		187	ｻ	219	ﾛ	251	
28	FS	60	<	92	¥	124	|	156		188	ｼ	220	ﾜ	252	
29	GS	61	=	93	]	125	}	157		189	ｽ	221	ﾝ	253	
30	RS	62	>	94	^	126	~	158		190	ｾ	222	ﾞ	254	
31	US	63	?	95	_	127	DEL	159	↓	191	ｿ	223	ﾟ	255	↓

INDEX

注.（ ）内は本文表記

あ

後付け条件……170

い

1行は255文字以内という制限……173
引用符「"」ではさまれた文字……38
引用符の中の（では）引用符……239

え

エディタ……54
エラーメッセージ……30
演算子……84, 226

お

親……138
親子関係……138
終わらない FOR 命令……165

か

カーソル……24
カーソルを移動する……238
下位節……138
仮想的な親……139
カッコ（ ）……32
カナロックキー……26
画面制御……238
画面を消す……238
空文字列……157
漢字処理機能……236
関数……226

き

強制中断……127
行……44
行の前後関係……45
行の（みを）表示……48
行ラベル……44
行を入れ替える……51
行を削除……49
行を挿入……52

く

空白文字……30
空白……30, 98
グローバル変数……109
グローバルリストユーティリティ……119

け こ

計算に優先順位がない……32
子供……138

し す せ そ

仕事のレベル……126
上位節……138
スペースバー……30
制御文字……75
整数除算……82
セミコロン……65
先頭に行を挿入する……53
添え字付き変数……87
添え字……88

ち て

注釈文……65
データベース……11, 220

に

入力指示……80

は ひ

パターン照合演算子……228
引数……41
引数のない WRITE 命令……75

ふ

「復帰改行」……40
プリンタに出力……251
プログラムリスタ……251
プロンプト……80

へ　ほ

変数……70
変数の親子関係……138
変数の兄弟……159
変数を指定しない KILL 命令……77
変数名……72
「法」……82
包含演算子……107

ま　め　も

「または」……234
命令語……170
メインドライバー……212
文字の色を変え（る）……238
モジュール……202
モジュロ……82
文字列添え字……177

ゆ

ユーティリティプログラム……62

り　る

リターンキー……24
ルーチン……33
ルーチンを登録……57, 65
ルーチンの（を）呼び出し……59, 120, 213
ルーチン名……60
ルーチンを削除する……61
ルーチンを（画面に）表示……47
ルーチンのリストをプリンタに印刷……251
ルーチン・ヘッダー……65

れ　ろ

連結演算子……228
ローカル変数……109

わ

ワークエリア……51

A

ATOK4……236

C

CAPS ロック・キー……26
CLOSE 命令……245
〈CMMND〉……30
CTRL/C……127

D

D ^ルーチン名……120
DO 命令……35

E　F　H

EGR98……241
FOR 命令……94
F QQ=0:0……165, 208
HALT 命令……27

I

IF 命令……100
〈INRPT〉……127
〈ISYTX〉……250

K　L

KILL 命令……76, 136
〈LINER〉……48

M

MS-DOS……22
MUMPS を停止する方法……27

〈MXSTR〉……247

N O P Q R

NEC の文節変換システム……236
〈NOPGM〉……62
OPEN 命令……244
〈PGMOV〉……77
QUIT 命令……74, 131
READ 命令……78

S

SET 命令……70
Simple Chart……244
〈SYNTX〉……30

T U

TAB キー……34
UCD-MUMPS……241
〈UNDEF〉……76
USE 命令……245

V

VJE-*a* および Σ……236

W

WRITE 命令……31
W ?10, X……97
W""……38
W!……40
W *X……239

X

XECUTE 命令……239

Z

ZLOAD 命令……58
ZPRINT 命令……35, 47
ZREMOVE 命令……34, 49
ZSAVE 命令……57

記 号

BS キー……26, 33
↵キー……24
$CHAR 関数……242
$DATA 関数……146
$EXTRACT 関数……227
$LENGTH 関数……226
$ORDER 関数……159, 184
$PIECE 関数……227
$RANDOM 関数……240
$SELECT 関数……232
$ZCALL 関数……248
$ZF（という）関数……243
D ^%GD……113
D ^%GL……119
D ^%RD……62
D ^%UPL……251
X ^%……54
X ^%UE……249
X ^%US……66
『?』記号……228
『=』記号……103
『^』記号……109, 120
『*』記号……31
『/』記号……31
『+』記号……230
『−』記号……228
『Y』記号……82
『>』記号……103
『[』記号……107
『<』記号……103
『!』記号……234

■参考文献

「標準MUMPS演習」 A. F. Krieg／L. K. Shearer 著
若井一朗訳
日刊工業新聞社
1983年

「MUG編／コンピュータ言語MUMPS入門」
R. F. Walters／J. Bowie／J. C. Wilcox 著
若井一朗訳
日刊工業新聞社
1980年

「標準MUMPS言語マニュアル」
MUMPS Development Committee 編
若井一朗／田中豊穂／嶋　芳成訳
コロナ社
1979年

「標準MUMPSプログラミング入門」〈はじめてプログラミングを学ぶ人に〉
井上高志／千葉純浩著
日本MUMPS・ユーザーズ・グループ
1980年

「標準MUMPSプログラミング入門」〈すでにMUMPS以外の言語を知っている人に〉
服部敏夫／野口　弘著
日本MUMPS・ユーザーズ・グループ
1980年

「American National Stanbard Information Systems
Programming Language MUMPS」(ANSI X11.1-1984)
MUMPS Development Committee 編
American National Standards Institute
1984年

上記の他，日本MUMPS・ユーザーズ・グループ（日本MUG）から様々な資料を入手可能です。

日本MUGの連絡先は次の通りです。

〒461 名古屋市東区代官町39番15号 ☎052-936-5660
マンプスシステム研究所
日本MUMPS・ユーザーズ・グループ事務局

著者略歴

1954年7月28日生まれ
1980年3月名古屋大学医学部卒業
現在，岐阜県厚生農業協同組合連合会久美愛病院勤務（内科医）
日本MUMPS・ユーザーズ・グループ運営委員／評議員
MUMPS開発委員会日本MUMPS・ユーザーズ・グループ代表委員

共訳「標準MUMPS言語マニュアル」コロナ社
訳「標準MUMPSポケットガイド」日本MUMPS・ユーザーズ・グループ

■SP-MUMPSの技術・販売サポートは下記の通りです
■社　名
（開発）住友電気工業株式会社システム事業部MEシステム課
（販売）住電システム株式会社応用システム部
■お問い合わせ先（両社共通）
（東京）〒107 東京都港区元赤坂1丁目3番12号
☎(03)423-5901(代表)
（大阪）〒530 大阪市北区堂島1丁目2番5号 ボルカートビル4階
☎(06)347-7511(代表)

国際的標準データベース言語
ビギニングMUMPS

定価はカバーに記載されております

昭和61年4月25日　初版印刷
昭和61年4月28日　初版発行

著　者　嶋　芳成
発行者　孫　正義
発行所　株式会社日本ソフトバンク
出　版　部
〒102 千代田区四番町2-1
☎03(261)4095
印　刷　奥村印刷株式会社
製作協力　株式会社ウッドブック

Printed in Japan
ISBN4-930795-30-3
落丁本，乱丁本はお取替えいたします。

SOFT BANK

定価2,600円